Institut National
Polytechnique de Lorraine

École Nationale Supérieure de Géologie

École doctorale RP2E

Advanced Visualization and Modeling of Tetrahedral Meshes

THÈSE

présentée et soutenue publiquement le 7 Avril 2006

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

Spécialité Géosciences

par

Tobias FRANK

Composition du jury

Rapporteurs :	Jean-Paul CHILÈS Heinz KONIETZKY Klaus SPITZER
Examinateurs :	Guillaume CAUMON Bruno LÉVY Broder MERKEL
Invité :	Amaury LEGAIT
Directeurs :	Jean-Laurent MALLET Helmut SCHAEBEN

Centre de Recherches Pétrographiques et Géochimiques
Laboratoire d'Infographie et d'Analyse de Données
Rue du Doyen Marcel Roubault - 54500 Vandœuvre

Advanced Visualization and Modeling of Tetrahedral Meshes

Von der Fakultät für Geowissenschaften, Geotechnik und Bergbau
der Technischen Universität Bergakademie Freiberg
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
doctor rerum naturalium
Dr. rer. nat.,

vorgelegt

von Diplom Ingenieur Tobias Frank
geboren am 30. September 1977 in Weiden i. d. Opf.

Gutachter: Prof. Dr. H. Konietzky (Freiberg)
Dr. B. Lévy (Nancy)
Prof. Dr. J.-L. Mallet (Nancy)
Prof. Dr. H. Schaeben (Freiberg)
Prof. Dr. K. Spitzer (Freiberg)

Tag der Verleihung: 7. April 2006

This document was typeset using LaTeX 2_ε

Acknowledgments

This work was performed in the G◯CAD research group. First of all, I want to thank the director of this thesis, Jean-Laurent Mallet. As founder of the G◯CAD project he guided this work over the last three years. His esprit and energy were a great source of inspiration. Further, I would like to thank the co-director of this thesis, Helmut Schaeben, for initiating this cotutelle project and for discussion and comments on this work.

I want to thank Jean-Jaques Royer, Christian Le Carlier de Veslud and Guillaume Caumon for discussion and comments. Many thanks to Monique Cugurno who always helped me whenever I got lost with French administration. I also want to mention my colleges at LIAD. We spent many a good time together in and out of the laboratory: Francois, David, Laurent L., Laurent S., Remi, Bruno, Laetitia, Laurent C., Emmanuel F., Luc, Pierre K., Sarah, Marc-Olivier and very special thanks to Emmanuel L. and Pierre M. for helping me survive the first weeks in France and Anne-Laure for improving my English.

During this research project gOcadians from Earth Decision supported me with inside knowledge and discussed my results. I owe thanks to Didier, Fabien, Richard, Laurent D. and Jean-Claude.

Special thanks to my friends and my family for supporting me the last three years, especially Corinna and Rosa.

This research work was performed in the frame of the G◯CAD research project. The companies and academic members of the G◯CAD consortium are hereby acknowledged for funding this thesis.

Contents

List of Figures

Introduction

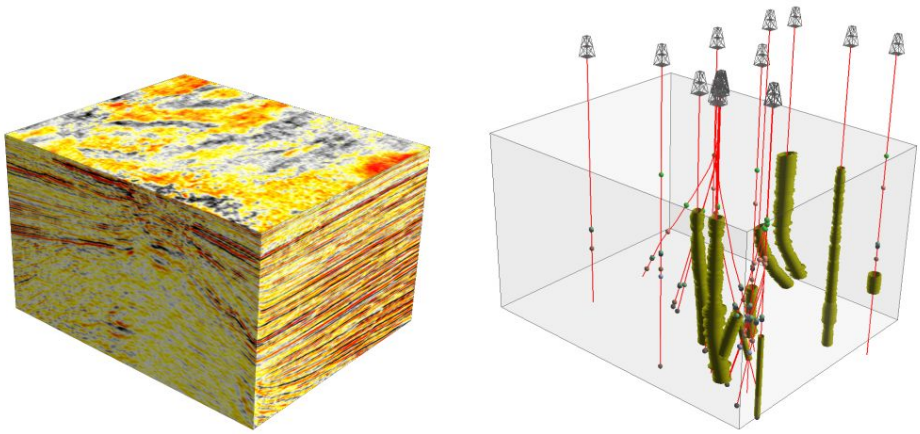
Geo-modeling deals with the numerical description of geometry and properties of the subsurface [Mallet, 2002]. Traditional Computer Aided Design (CAD) applications based on parametric representation of the objects are not adapted to model arbitrarily complex objects as they occur in natural sciences like geology. In the late eighties Professor Jean-Laurent Mallet proposed a discrete modeling approach realized in the GOCAD software. Discrete models can represent models of arbitrary complexity and this work is exclusively based on volumetric $3d$-discrete representation. Today GOCAD is an industrial software suite developed and maintained by Earth Decision (Houston, Texas). The GOCAD software suite is one of the most sophisticated geo-modeling tools that is widely used by the oil and gas industry as well as by numerous national and international geological surveys and universities. Structured grids like Cartesian or curvilinear grids are state-of-the-art for volumetric representation of $3d$-geological objects. For this type of grids a set of advanced visualization tools is available. However, unstructured grids like tetrahedral meshes are used more and more for geophysics and reservoir modeling. This trend asks for efficient methods for visualizing and modeling large data sets on tetrahedral meshes. The main objectives of this work are:

1. Visualization of geological information of tetrahedral meshes
2. Geometric and implicit modeling of tetrahedral meshes

The developed methods were implemented as a GOCAD plug-in using C++, OpenGL and CG. Due to the inconsistency of programmability of the graphics hardware of UNIX workstations, the developed plug-in is limited to Windows 2000/XP and Linux. The presented work was performed in the GOCAD research group as *Cotutelle de These* between the *Ecole Nationale Supérieure de Géologie* at *Institut National Polytechnique de Lorraine* (France) and the *Institute for Geoscience Mathematics and Informatics* at *Technische Universität Bergakademie Freiberg* (Germany).

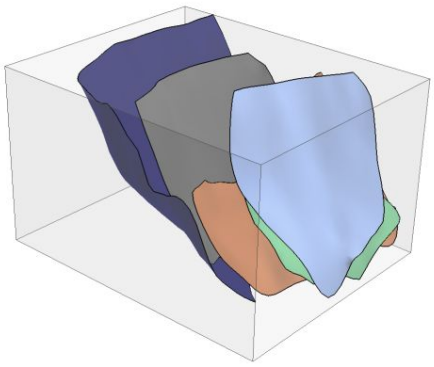
Commonly Used $3d$-Meshes for Geo-Modeling

There exists a wide range of techniques to model the subsurface. Starting from seismic data (see figure 1(a)) and well logs (see figure 1(b)), a so-called structural model (see figure 1(c) and 2(a)) is built. A structural model defines a bounding box and the

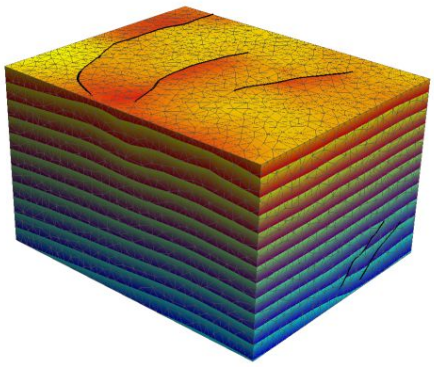


(a) Seismic

(b) Well logs

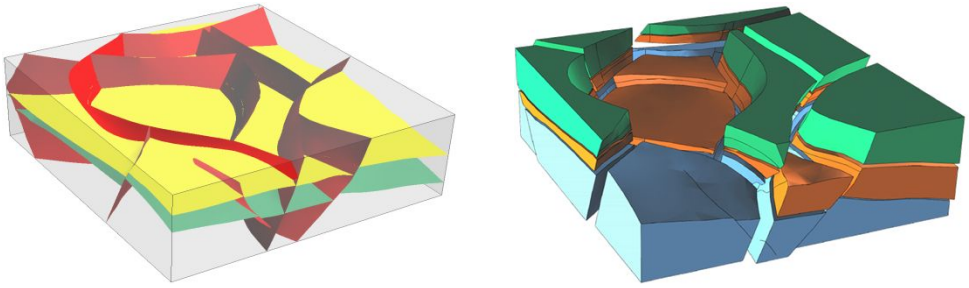


(c) Structural model



(d) Tetrahedralized volumetric model

Figure 1: Geophysical data like seismic are acquired in $2d$ or $3d$-Cartesian grids (a). Together with information from well data (b), a so-called structural model (c) is built. The topological information of the structural model is used to compute a cellular decomposition honoring the geological interfaces like faults (d). Data by courtesy of Earth Decision.



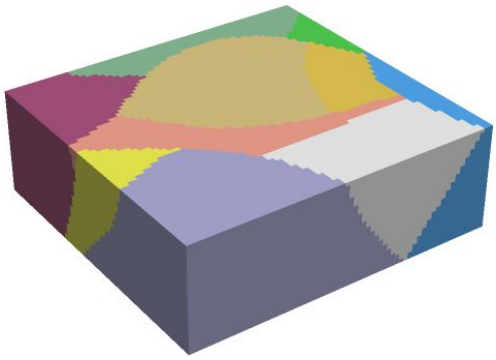
(a) Structural model (b) Boundary representation

Figure 2: (a) shows a structural model with faults and horizons. (b) shows the boundary representation of the regions defined by the structural model of (a). Data by courtesy of Total.

geological interfaces such as faults and horizons. Their mutual contacts define a topological model of the subsurface (see figure 2). Using such a boundary representation, analytical functions can be used to describe properties in these pre-defined volumes. This type of representation has been widely used in the early days of numerical modeling of $3d$-objects. However, they lead to many limitations especially for quantitative modeling like flow simulation (e.g. reservoir flow modeling) or volumetric simulations (e.g. geostatistic) [Caumon, 2003]. A solution to these problems is to create a discrete cellular decomposition of the regions of the considered model. Such a decomposition is defined by a mesh. One can distinguish two main types of meshes: structured and unstructured.

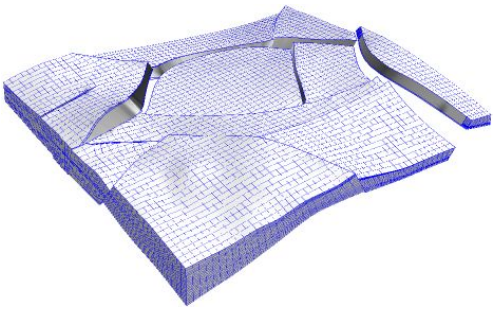
Structured meshes are characterized by a periodic pattern of the node connection. Each cell can be identified using an indexing scheme defined by the mesh itself. This indexing mechanism provides a rapid access to cells or nodes, respectively, in the neighborhood of a cell. This implicit addressing enables rapid access and very efficient memory management for attached properties. Figure 3(a) shows a decomposition into regular unit cubes. Such a Cartesian grid shows strong aliasing artifacts along curved boundaries (see figure 3(b)). These artifacts can be reduced using a higher resolution. But the number of cells can easily excess the computational possibilities. The so-called stratigraphic grids – that are often used to model stratigraphic sequences in reservoirs – are an alternative to reduce aliasing of Cartesian grids. Stratigraphic grids (see figure 3(c)) honor curvilinear geometry in a more precise way. They use hexahedrons with adaptive size and shape to describe complex deformed structures. As their cell size can be adapted to take complex shapes into account, stratigraphic grids are more efficient in terms of resolution than Cartesian grids. However, they still produce distortions in regions of multiple contacts (see figure 3(d)). Stratigraphic grids sometimes use workarounds like dead cells (cells with zero volume) to obtain a precise enough model. These dead cells are the major drawback for numerical methods like flow simulations.

A mesh which does not exhibit regularities in the node connection is called unstructured. If the mesh is constituted of cells with one distinct type of polyhedral shape, the mesh is homogeneous. If the mesh is composed of cells with different polyhedral shapes, the mesh is heterogenous. Visualization tools for heterogenous unstructured meshes were developed by Lévy et al. [2001] and Caumon et al. [2005], but this type of meshes are

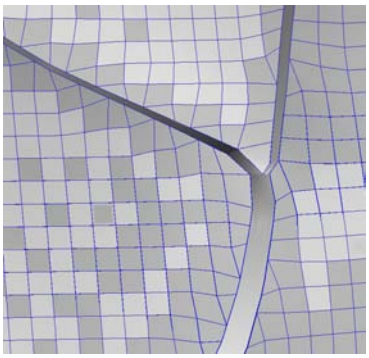


(a) Cartesian grid: 131,072 voxels

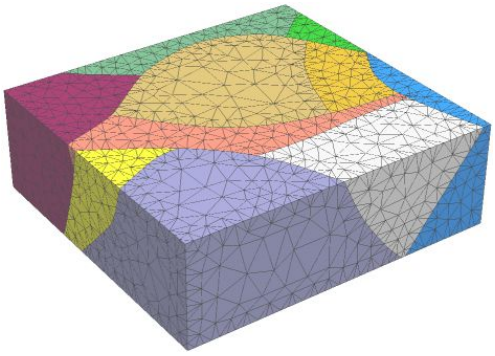
(b) Aliasing effects on volume boundaries non-parallel to the volume axis.



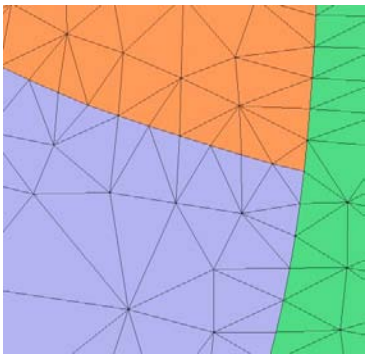
(c) Stratigraphic grid: 52,000 hexahedrons



(d) Distorted hexahedrons to honor geometrically complex zones with multiple contacts.



(e) $3d$-simplicial complex: 12,879 tetrahedrons



(f) Tetrahedrons honor geometry of arbitrary complexity without artifacts.

Figure 3: This series of figures shows a cellular decomposition of a geological model. Structured meshes (a) and (c) show aliasing effects on curved region boundaries (b) and distortions in regions with multiple contacts (d). Unstructured tetrahedral meshes (e) can model geometry of arbitrary complexity and eliminate the artifacts of structured meshes (f). Data by courtesy of Total.

not fully implemented in the G◯CAD framework. This work deals with homogenous unstructured tetrahedral meshes. This type of mesh is also called $3d$-simplicial complex. Figure 3(e) shows an example of a tetrahedral mesh. Geometry of any complexity can be modeled by this type of mesh and hereby produces no artifacts on multiple contacts (see figure 3(f)). Unstructured meshes can also define an adaptive resolution with small cells in regions with complex geometry or where a high numerical resolution is required. The same model can be defined by a smaller amount of cells and offers a higher precision compared to structured meshes (see figure 3).

Tetrahedral Meshes for Geo-Modeling Applications

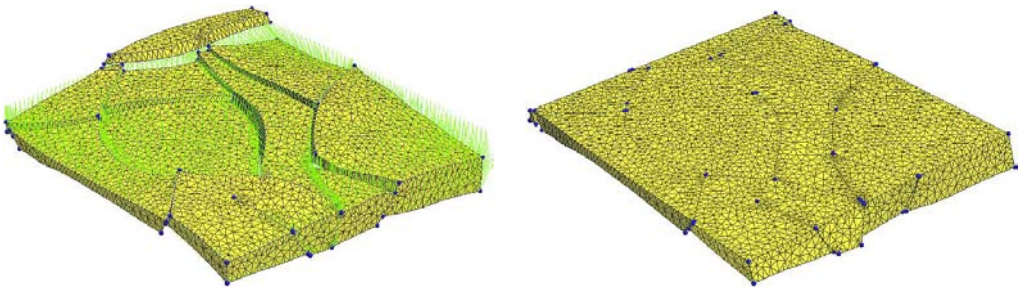
Tetrahedral meshes play an important role in finite differences, finite volume or finite element methods, which are extremely used in scientific computing such as mechanics, flow simulation, biology and many more. Advances in tetrahedral mesh generation methods honoring various conditions encountered in subsurface modeling [Lepage, 2003] led to an increasing application of tetrahedral meshes in recent research projects related to geo-modeling. The following sections shortly introduce recent research projects based on tetrahedral meshes for geo-modeling applications.

Reservoir Simulation

Reservoir simulations are mainly performed on Cartesian and stratigraphic grids. As mentioned before, these grids cannot represent complex geometries like fault contacts. Prévost et al. [2001] extended the Pollock streamline tracing method previously developed for Cartesian grids [Pollock, 1988] to unstructured tetrahedral meshes. Streamlines are an important tool to estimate reservoir performance since they provide a fast characterization of flows in reservoirs. The methodology used to build a streamline in a tetrahedron is to decompose each tetrahedron into a set of subcells defined by the edge-midpoints and its barycenter. The streamline is computed locally on the subcells. The average fluxes in the subcells are used to compute the streamline trace in a tetrahedron.

Restoration

Subsurface models are built from seismic surveys and well data. The vertical resolution of seismic data is generally poor, leading to a lack of precision, while wells are sparse and irregularly distributed. Thus, it is important to assure the consistency of the model and to post-validate the various assumptions made along the modeling process; especially when the geological formations have been faulted. One tries to verify that the deformations contained in the present model are consistent with their shapes at the time of their formation (deposition). The model is considered as an incompressible material and each particle of this material is displaced during the restoration to its original position. $3d$-structural restoration is performed on tetrahedral meshes. A restoration vector is computed at each node of the tetrahedral mesh (see figure 4(a)). The restoration vectors must honor two



(a) Initial model with restoration vectors (b) Restored model

Figure 4: These figures illustrate a $3d$-structural restoration performed on a tetrahedral mesh. For each node a restoration vector is computed (a). The restored model (b) is obtained by displacing the nodes along their restoration vectors [Muron, 2005]. Data by courtesy of Total.

types of constraints: the mass balance equations and minimal strain tensor [Mallet, 2002]. Figure 4(b) shows the restored model after displacing the nodes of the tetrahedral mesh according to the restoration vectors [Muron, 2005].

Fractured Reservoirs

Natural fractures play an important role for hydrocarbon recovery as they can dominate the hydraulic flow. The characterization of a naturally fractured reservoir is needed for a reliable flow model. Assuming a stochastic model, fracturation can be considered as random events. Macé et al. [2005] compute the probability distribution of the density (see figure 5) and orientation of fractures on a tetrahedral mesh. The $3d$-strain field is an input parameter obtained by $3d$-balanced restoration [Muron, 2005] or GeoChron parametrization [Mallet, 2004; Mallet et al., 2004; Moyen, 2005]. Both methods extensively use tetrahedral meshes.

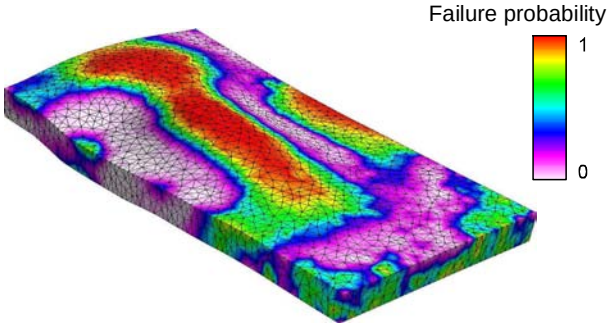
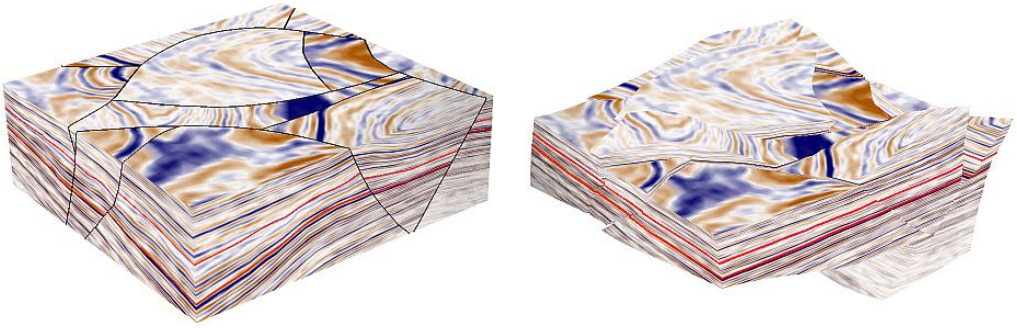


Figure 5: This figure shows the fracture probability computed on a tetrahedral mesh [Macé et al., 2005]. Data by courtesy of Institut Français du Pétrole (IFP).

GeoChron

Mallet [2004] introduced the GeoChron parametrization for sedimentary geology. In this parametric space the model is flattened so that all horizons are planes (see figure 6). The GeoChron parametrization is computed on tetrahedral meshes [Moyen, 2005]. More information on the GeoChron parametrization can be found in section 1.2 of this work.



(a) Geological space (b) GeoChron space

Figure 6: (a) shows a model in the geological space rendered with seismic amplitude. The faults are highlighted with black outlines and the reference horizons with red outlines. (b) shows the same model in the parametric GeoChron space. The faults have disappeared and the horizons are horizontal planes. Using the GeoChron parametrization, seismic cubes can be flattened. Data by courtesy of Total.

Organization of this Work

The previous section showed recent research projects that enlighten the importance of tetrahedral meshes in various fields of geo-modeling. Tetrahedral meshes were developed from the early days of the GOCAD project [Conraud, 1997; Lepage, 2003]. So far, the tetrahedral mesh was a neglected object inside GOCAD. But recent applications use tetrahedral meshes with complex geometry and topology. Therefore advanced tools for visualization and modeling of this type of mesh are essential. This work introduces tools for *advanced visualization and modeling of tetrahedral meshes*. The visualization techniques were adapted for the needs of geological applications and special care was taken for future extensions. Modeling can be subdivided into *geometric* and *implicit* modeling. Geometric modeling deals with the densification of a given tetrahedral mesh to honor the numerical resolution of input data and zones of certain interest. Implicit modeling covers a real $3d$-approach to create geological interfaces like salt-top surfaces or horizons from noisy point sets and the interactive real-time manipulation of such interfaces. This thesis is organized as follows:

Chapter 1 introduces the theoretical background: especially a matrix formulation of the DSI [Mallet, 1992, 2002, 2003] method for tetrahedral meshes and the GeoChron [Mallet, 2004; Mallet et al., 2004; Moyen, 2005] parametrization. This chapter also defines terms and notations used throughout this work.

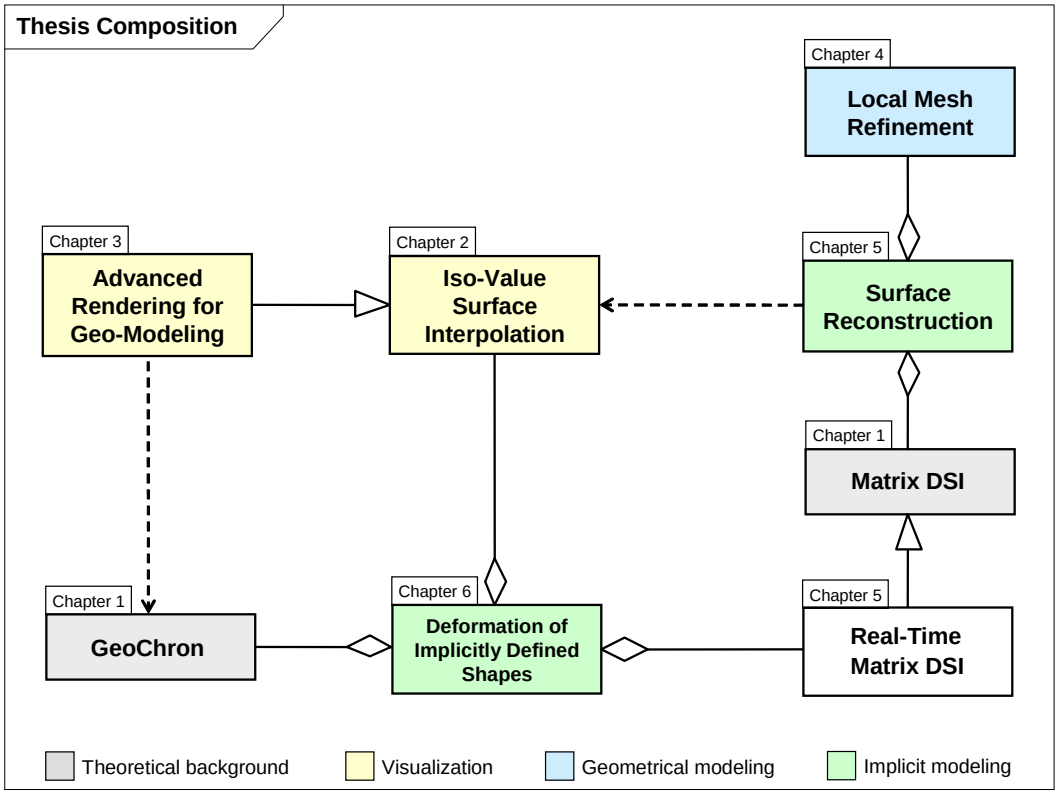


Figure 7: This UML diagram illustrates the dependencies between the different components of this thesis and its theoretical background. Especially modeling depends on the visualization tools and implicit modeling also makes use of geometric modeling to improve numerical accuracy.

Chapter 2 represents numerical methods developed for iso-value surface interpolation of $3d$-scalar fields defined on tetrahedral meshes. Implementation issues and parallelization for modern hardware architectures are covered.

Chapter 3 extends the basic visualization with advanced features like the computation and visualization of distance maps or stratigraphic columns on unstructured grids. Co-rendering of multiple attributes as well as boolean operations of Constructive Solid Geometry performed by the graphics hardware is demonstrated.

Chapter 4 compares different mesh refinement methods for tetrahedral meshes. Its objectives are to improve the numerical resolutions of applications based on the DSI method. Mesh refinement is later largely used for surface reconstruction.

Chapter 5 introduces an automatic method to reconstruct multi-valued geological interfaces like complex salt-top surfaces from noisy point clouds. An implicit representation of the interface is computed on a tetrahedral mesh.

Chapter 6 illustrates an interactive method for the real-time manipulation of implicitly defined shapes like geological horizons. A real-time extension of the DSI method is introduced.

The chapters are not independent from one another. Especially the modeling depends on the visualization tools and implicit modeling uses geometric modeling to improve the numerical accuracy. Figure 7 illustrates these relationships between the different subjects.

Chapter 1

Theoretical Background

Contents

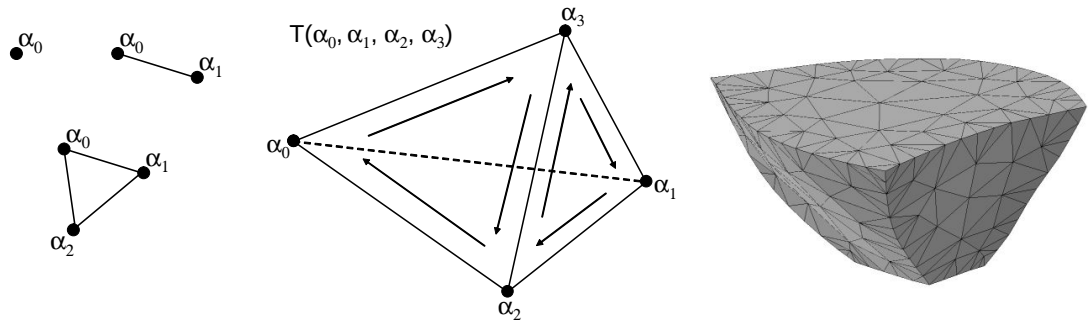
1.1 Simplicial Complex

Tetrahedral meshes are also called *simplicial complexes*. A *n-simplex* is a n-dimensional polytype[1] consisting of $n+1$ vertices and is the convex hull in the $\mathbb{R}^n$ Euclidean space. Any vertex is connected to the remaining vertices by an edge. A $0d$-simplex is a point, a $1d$-simplex a line segment, a $2d$-simplex a triangle and a $3d$-simplex a tetrahedron. An aggregation of n-dimensional cells to a n-dimensional composite is called complex (see figure 1.1). A n-dimensional complex that consists only of n-dimensional simplices is called a simplicial complex (see figure 1.1). The simplicial complex considered in this work is a discrete linear model embedded into the $3d$-Euclidean space that consists of a set Ω of interconnected nodes on which the values of a function $\varphi(x,y,z)$ are defined. The function's values are interpolated linearly between the nodes. An iso-value surface $\mathscr{S}_\phi$ is defined by a level set were $\varphi(x,y,z)$ takes a constant value ϕ so that $\varphi - \phi = 0$.

1.1.1 Notion of a Linear Function on a Tetrahedron

The linear tetrahedron method was investigated by numerous authors (e.g. [Hinton and Owen, 1979] and [Shewchuk, 2002]). The formulation proposed by Mallet [2003] is adopted in the following. Each tetrahedron $T = T(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ of the domain is embedded into the $3d$-Euclidean space. The vertices $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$ of T carry the values $\{\varphi(\alpha_0), \varphi(\alpha_1), \varphi(\alpha_2), \varphi(\alpha_3)\}$ of the property $\varphi(x,y,z)$. Assuming that the faces of T are

[1] A polytype is the generalization of a $2d$-polygon to n dimensions.



(a) $0d$, $1d$, $2d$-simplex (b) $3d$-simplex (c) $3d$-simplicial complex (709 simplices)

Figure 1.1: This series of figures shows simplices in different dimensions (a) and (b) and the composition of tetrahedrons to a $3d$-simplicial complex (c). The edges and faces of a tetrahedron store references to the adjacent cells.

plane and that the function $\varphi(x,y,z)$ varies linearly inside T, the property $\varphi(x,y,z)$ can be represented for any point $(x,y,z) \in T$ as a linear function:

$$\varphi(x,y,z) = [1,x,y,z] \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{1.1}$$

The coefficients $\{a_0,a_1,a_2,a_3\}$ depend both on the values $\{\varphi(\alpha_0),\varphi(\alpha_1),\varphi(\alpha_2),\varphi(\alpha_3)\}$ taken by $\varphi(x,y,z)$ at the nodes of T and on the geometry of T. Let (x_i,y_i,z_i) be the coordinates of the nodes α_i $(i = 0,1,2,3)$ of a tetrahedron T, so the coefficients $\{a_0,a_1,a_2,a_3\}$ are the solution of the following linear system.

$$\underbrace{\begin{bmatrix} 1 & x_0 & y_0 & z_0 \\ 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \end{bmatrix}}_{M} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \varphi(\alpha_0) \\ \varphi(\alpha_1) \\ \varphi(\alpha_2) \\ \varphi(\alpha_3) \end{bmatrix} \tag{1.2}$$

As a consequence, the values of the implicit function φ can be computed at any location (x^*,y^*,z^*) inside T as function of $\{\varphi(\alpha_0),\varphi(\alpha_1),\varphi(\alpha_2),\varphi(\alpha_3)\}$ at the nodes of T:

$$\varphi(x^*,y^*,z^*) = \underbrace{[1,x^*,y^*,z^*] \cdot M^{-1}}_{[b_0,b_1,b_2,b_3]} \cdot \begin{bmatrix} \varphi(\alpha_0) \\ \varphi(\alpha_1) \\ \varphi(\alpha_2) \\ \varphi(\alpha_3) \end{bmatrix} \tag{1.3}$$

Provided the matrix M is invertible which is the case for non-degenerated[2] tetrahedrons. The coefficients $\{[b_0,b_1,b_2,b_3]\}$ are called the barycentric coordinates of a point

[2] A degenerated tetrahedron is a $3d$-simplex with its four vertices coincident with a plane, a line or single point.

(x^*, y^*, z^*) inside T relative to the nodes $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$. The coefficients $D_x(\alpha_i)$, $D_y(\alpha_i)$, $D_z(\alpha_i)$ are defined by:

$$\begin{bmatrix} D_x(\alpha_0) & D_x(\alpha_1) & D_x(\alpha_2) & D_x(\alpha_3) \\ D_y(\alpha_0) & D_y(\alpha_1) & D_y(\alpha_2) & D_y(\alpha_3) \\ D_z(\alpha_0) & D_z(\alpha_1) & D_z(\alpha_2) & D_z(\alpha_3) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot M^{-1} \tag{1.4}$$

According to (1.3), the derivatives of φ are constant inside T and can be written as linear combinations of the values $\{\varphi(\alpha_0), \varphi(\alpha_1), \varphi(\alpha_2), \varphi(\alpha_3)\}$:

$$\begin{aligned} \frac{\partial \varphi}{\partial x} &= \sum_{i=0}^{3} D_x(\alpha_i) \cdot \varphi(\alpha_i) \\ \frac{\partial \varphi}{\partial y} &= \sum_{i=0}^{3} D_y(\alpha_i) \cdot \varphi(\alpha_i) \\ \frac{\partial \varphi}{\partial z} &= \sum_{i=0}^{3} D_z(\alpha_i) \cdot \varphi(\alpha_i) \end{aligned} \tag{1.5}$$

Let m be defined as the following square matrix:

$$m = \begin{bmatrix} (x_1 - x_0) & (y_1 - y_0) & (z_1 - z_0) \\ (x_2 - x_0) & (y_2 - y_0) & (z_2 - z_0) \\ (x_3 - x_0) & (y_3 - y_0) & (z_3 - z_0) \end{bmatrix} \tag{1.6}$$

It can be verified that:

$$m^{-1} \cdot \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot M^{-1} \tag{1.7}$$

thus, the coefficients $D_x(\alpha_i)$, $D_y(\alpha_i)$, $D_z(\alpha_i)$ defined in equation (1.4) can also be computed as follows:

$$\begin{bmatrix} D_x(\alpha_0) & D_x(\alpha_1) & D_x(\alpha_2) & D_x(\alpha_3) \\ D_y(\alpha_0) & D_y(\alpha_1) & D_y(\alpha_2) & D_y(\alpha_3) \\ D_z(\alpha_0) & D_z(\alpha_1) & D_z(\alpha_2) & D_z(\alpha_3) \end{bmatrix} = m^{-1} \cdot \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \tag{1.8}$$

To simplify the next formulas let the vectors $\mathbf{D}(\alpha_i)$ be defined by:

$$\mathbf{D}(\alpha_i) = \begin{bmatrix} D_x(\alpha_i) \\ D_y(\alpha_i) \\ D_z(\alpha_i) \end{bmatrix} \quad \forall \ i = 0, 1, 2, 3 \tag{1.9}$$

Substituting in equation (1.8) leads to:

$$[\mathbf{D}(\alpha_0), \mathbf{D}(\alpha_1), \mathbf{D}(\alpha_2), \mathbf{D}(\alpha_3)] = m^{-1} \cdot \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

By assumption, the property $\varphi(x,y,z)$ is interpolated linearly inside each tetrahedron T, so the gradient $\nabla\varphi$ is constant inside T. From equations (1.5) and (1.8), $\nabla\varphi$ can be expressed as follows:

$$\nabla\varphi = \begin{bmatrix} \partial\varphi/\partial x \\ \partial\varphi/\partial y \\ \partial\varphi/\partial z \end{bmatrix} = \sum_{i=0}^{3} \mathbf{D}(\alpha_i)\cdot\varphi(\alpha_i) \tag{1.11}$$

A scalar product of the gradient $\nabla\varphi$ with a vector $\mathbf{w}$ can then be represented as a linear combination of the scalar coefficients $D(\alpha_i|\mathbf{w})$:

$$\begin{aligned} \nabla\varphi\cdot\mathbf{w} &= \sum_{i=0}^{3} D(\alpha_i|\mathbf{w})\cdot\varphi(\alpha_i) \quad \text{with:} \\ D(\alpha_i|\mathbf{w}) &= \mathbf{D}(\alpha_i)\cdot\mathbf{w} \quad \forall\ i=0,1,2,3 \end{aligned} \tag{1.12}$$

More information on the linear tetrahedral model and its extension to degenerated tetrahedrons can be found in [Mallet, 2003] and [Royer, 2005].

1.1.2 Matrix DSI on *3d*-Tetrahedral Meshes

This section introduces the Discrete Smooth Interpolation (DSI) [Mallet, 1992, 2002, 2003; Muron et al., 2005] method to interpolate a function on the nodes of a discrete tetrahedral mesh. For the sake of simplicity, this presentation is limited to scalar functions. Let $\mathscr{M}(\Omega,N,\varphi,\mathscr{C})$ be a discrete model defined in the $3d$-Euclidean space. A finite set Ω of n interconnected nodes defines the tetrahedral mesh that can also be considered as a connected graph. This graph defines a neighborhood operator N on Ω so that the neighborhood $N(\alpha)$ of any node $\alpha\in\Omega$ is a subset of nodes of Ω which are directly connected to the node α plus α itself. A function φ on Ω is defined as follows:

$$\varphi = \begin{bmatrix} \varphi(1) \\ \vdots \\ \varphi(\alpha) \\ \vdots \\ \varphi(n) \end{bmatrix} \begin{matrix} \uparrow \\ \vdots \\ (n) \\ \vdots \\ \downarrow \end{matrix} \tag{1.13}$$

The last component $\mathscr{C}$ of the linear discrete model is a set of linear constraints which have to be honored by φ:

$$A_c^t\cdot\varphi \bowtie b_c \quad \forall c\in\mathscr{C} \tag{1.14}$$

where A_c is a given column matrix of the same size as φ, b_c a given coefficient and $\bowtie$ one of the three operators $\{\simeq,=,\geq\}$.

DSI Constraints, Overview

The binary operator $\bowtie$ defined in equation 1.14 fully characterizes the possible constraint classes. In general, two sets of constraints can be defined: soft constraints $\mathscr{C}^s$ such as $\simeq$

and hard constraints $\mathscr{C}^h$ $(=,\geq)$.

$$\mathscr{C} = \mathscr{C}^s \cup \mathscr{C}^h \text{with:} \left| \begin{array}{rcl} c \in \mathscr{C}^s & \Leftrightarrow & A_c^t \cdot \varphi \simeq b_c \\ c \in \mathscr{C}^h & \Leftrightarrow & \left\{ \begin{array}{c} A_c^t \cdot \varphi = b_c \\ or \\ A_c^t \cdot \varphi \geq b_c \end{array} \right. \end{array} \right. \tag{1.15}$$

The DSI method is an interpolator that approximates the function φ so that the hard constraints $\mathscr{C}^h$ are strictly honored and the soft constraints $\mathscr{C}^s$ are honored in a least squares sense. Additionally, there must always exist a subset of soft constraints $\mathscr{C}^{ss}$, which define a smoothness criterion. Smoothness criterion means that the function $\varphi(\alpha)$ varies smoothly from one location α to its neighbor nodes $\beta \in N(\alpha)$. Thus, it does not necessarily fit the observations strictly at location α (kind of moving average). By definition of the DSI method, a solution only exists if the hard constraints $\mathscr{C}^h$ are consistent. It is important to note that the consistence of soft constraints is not mandatory because they are honored in a least squares sense. The column matrix A_c has the same size as the column matrix of the implicit function φ.

$$A_c = \left[\begin{array}{c} A_c(1) \\ \vdots \\ A_c(\alpha) \\ \vdots \\ A_c(n) \end{array} \right] \begin{array}{c} \uparrow \\ \vdots \\ (n) \\ \vdots \\ \downarrow \end{array} \tag{1.16}$$

DSI Method

Let us assume that the set of hard constraints $\mathscr{C}^h$ is empty in a first approach. So the set of DSI constraints $\mathscr{C}$ consists of k soft constraints $\mathscr{C} = \mathscr{C}^s = \{c_1, ..., c_k\}$. The DSI method provides k linear equations that reflect the set of soft constraints $\mathscr{C}^s$.

$$\left| \begin{array}{rcl} A_{c1}^t \cdot \varphi & \simeq & b_{c1} \\ & \vdots & \\ A_{ck}^t \cdot \varphi & \simeq & b_{ck} \end{array} \right. \tag{1.17}$$

The DSI interpolator minimizes the interpolation error induced by the violation of soft constraints in a least square sense. This problem is equivalent to minimizing the least squares criterion (LS):

$$LS(\varphi) = \sum_{c \in \mathscr{C}^s} \left\| A_c^t \cdot \varphi - b_c \right\|^2 \tag{1.18}$$

$LS(\varphi)$ is a global criterion for the violation of the soft constraints in $\mathscr{C}$ and $\|\cdot\|$ is the Euclidean norm. Additionally, for each constraint a weighting factor can be applied by transforming A_c and b_c linearly as follows:

$$\forall\ c \in \mathscr{C}^s : \left| \begin{array}{rcl} A_c & \leftarrow & \frac{\varpi_c}{\|A_c\|} \cdot A_c \\ b_c & \leftarrow & \frac{\varpi_c}{\|A_c\|} \cdot b_c \end{array} \right. \tag{1.19}$$

The effect of this normalization is that each constraint $c \in \mathscr{C}^s$ now has a relative weight ϖ_c^2 compared to the other constraints of $\mathscr{C}^s$. This coefficient ϖ_c is also called the certainty factor for constraint c. The right hand side of equation (1.18) can be developed as follows:

$$\begin{aligned} \left\| A_c^t \cdot \varphi - b_c \right\|^2 &= (A_c^t \cdot \varphi - b_c)^t \cdot (A_c^t \cdot \varphi - b_c) \\ &= \varphi^t \cdot (A_c \cdot A_c^t) \cdot \varphi - 2 \cdot (b_c \cdot A_c^t) \cdot \varphi + b_c^2 \end{aligned} \tag{1.20}$$

Introducing a square symmetrical matrix $\mathbf{A}$ and a vector $\mathbf{b}$:

$$\left| \begin{array}{lcl} \mathbf{A} &=& \sum_{c \in \mathscr{C}^s} A_c \cdot A_c^t \\ \mathbf{b} &=& \sum_{c \in \mathscr{C}^s} b_c \cdot A_c \end{array} \right. \tag{1.21}$$

and applying equation (1.20) to (1.18) the LS (1.18) can be simplified to a LS defined by:

$$LS(\varphi) = \frac{1}{2} \cdot \varphi^t \cdot \mathbf{A} \cdot \varphi - \mathbf{b}^t \cdot \varphi \tag{1.22}$$

The quadric form is minimized using the Conjugate Gradient (CG) method [Hestenes and Stiefel, 1952; Shewchuk, 1994]. Starting with an arbitrary solution φ^0, the Conjugate Gradient method requires in most cases less than n steps for symmetrical, positive definite matrices, such as $\mathbf{A}$. Note that if the initial solution is close to the final result, then the Conjugate Gradient method converges much faster than in n steps.

Control Point Constraint (Input Data)

At a given data point (x^*, y^*, z^*) inside a tetrahedron $T(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ of a $3d$-simplicial complex, the value of the implicit function $\varphi(x^*, y^*, z^*)$ should be equal to a given value ϕ. Using equation (1.3) it comes:

$$\phi = \underbrace{[1, x^*, y^*, z^*] \cdot M^{-1}}_{[b_0, b_1, b_2, b_3]} \cdot \begin{bmatrix} \varphi(\alpha_0) \\ \varphi(\alpha_1) \\ \varphi(\alpha_2) \\ \varphi(\alpha_3) \end{bmatrix} \tag{1.23}$$

The control point constraint $c = c(T, x^*, y^*, z^*)$ consists in specifying that $\varphi(x^*, y^*, z^*) = \phi$ and leading to the following values for A_c and b_c:

$$c : \left| \begin{array}{lcll} A_c(\alpha_i) &=& b_i & \text{if } \alpha_i \in \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} \\ A_c(\alpha_i) &=& 0 & \text{otherwise} \\ b_c &=& \phi & \end{array} \right. \tag{1.24}$$

Constant Gradient Constraint (Roughness)

Let $T^\circ = T(\alpha_0^\circ, \alpha_1, \alpha_2, \alpha_3)$ and $T^* = T(\alpha_0^*, \alpha_1, \alpha_2, \alpha_3)$ be two adjacent tetrahedrons sharing three common nodes $\{\alpha_1, \alpha_2, \alpha_3\}$ of the triangular face $t = t(\alpha_1, \alpha_2, \alpha_3)$. Let $\mathbf{n}$ be the unit vector that is orthogonal to this common triangle. The constant gradient constraint attempts to constrain the scalar piecewise linear implicit function $\varphi(x, y, z)$ to have the

same gradient inside T° and T^*. This constraint is used as smoothness criterion and it is installed as soft constraint in $\mathscr{C}^{ss}$. This is due to the fact that φ is linearly interpolated inside a tetrahedron, so its gradient is constant and cannot be equal – for non-constant functions – when crossing a common triangular face. The gradient can only be "equal" in a least square sense, i.e. the gradients are varying smoothly from one tetrahedron to its neighbors. By definition $\nabla^\circ\varphi$ and $\nabla^*\varphi$ are the gradients of the function $\varphi(x,y,z)$ inside the tetrahedrons T° and T^*. We define $proj(\mathbf{w})_t$ as projection of any vector $\mathbf{w}$ of the $3d$-Euclidean space onto the plane containing the triangle t:

$$\left|\begin{array}{l}\nabla^\circ\varphi = proj(\nabla^\circ\varphi)_t + (\mathbf{n}\cdot\nabla^\circ\varphi)\cdot\mathbf{n}\\ \nabla^*\varphi = proj(\nabla^*\varphi)_t + (\mathbf{n}\cdot\nabla^*\varphi)\cdot\mathbf{n}\end{array}\right. \tag{1.25}$$

By definition of a piecewise linear function, the variation of φ inside a tetrahedron is linear. So the projections $proj(\nabla^\circ\varphi)_t$ and $proj(\nabla^*\varphi)_t$ only depend linearly on the values $\varphi(\alpha_1)$, $\varphi(\alpha_2)$ and $\varphi(\alpha_3)$. The gradients of φ in T° and T^* are equal only if their scalar product with the normal vector $\mathbf{n}$ are identical:

$$\mathbf{n}\cdot\nabla^\circ\varphi = \mathbf{n}\cdot\nabla^*\varphi \tag{1.26}$$

Using the coefficients $D^\circ(\alpha_i|\mathbf{n})$ and $D^*(\alpha_i|\mathbf{n})$ defined by equation (1.12):

$$\left|\begin{array}{l}D^\circ(\alpha_i|\mathbf{n}) = \mathbf{D}^\circ(\alpha_i)\cdot\mathbf{n}\ \ \forall\alpha_i\in\{\alpha_0^\circ,\alpha_1,\alpha_2,\alpha_3\}\\ D^*(\alpha_i|\mathbf{n}) = \mathbf{D}^*(\alpha_i)\cdot\mathbf{n}\ \ \forall\alpha_i\in\{\alpha_0^*,\alpha_1,\alpha_2,\alpha_3\}\end{array}\right. \tag{1.27}$$

and by substitution in equation (1.26) it comes:

$$D^\circ(\alpha_0|\mathbf{n})\cdot\varphi(\alpha_0^\circ)+\sum_{i=1}^{3}D^\circ(\alpha_i|\mathbf{n})\cdot\varphi(\alpha_i) = D^*(\alpha_0|\mathbf{n})\cdot\varphi(\alpha_0^*)+\sum_{i=1}^{3}D^*(\alpha_i|\mathbf{n})\cdot\varphi(\alpha_i) \tag{1.28}$$

Equation 1.28 provides a linear relationship between $\varphi(\alpha)$ and its neighbors such that the gradient is constant in T° and T^*. It can be turned into a DSI constraint $c = c(T^\circ,T^*)$ with the following coefficients:

$$c:\left|\begin{array}{rcll}A_c(\alpha_0^\circ) &=& D^\circ(\alpha_0^\circ|\mathbf{n}) &\\ A_c(\alpha_0^*) &=& -D^*(\alpha_0^*|\mathbf{n}) &\\ A_c(\alpha_i) &=& D^\circ(\alpha_i|\mathbf{n})-D^*(\alpha_i|\mathbf{n}) & \text{if } \alpha_i\in\{\alpha_1,\alpha_3,\alpha_3\}\\ A_c(\alpha) &=& 0 & \text{otherwise}\\ b_c &=& 0 &\end{array}\right. \tag{1.29}$$

Such a constraint must be applied for any pair of topologically adjacent tetrahedrons in order to minimize the "roughness" of the function φ.

1.2 GeoChron

This work uses a second technology: GeoChron [Mallet, 2004; Mallet et al., 2004; Moyen, 2005]. The basic idea behind GeoChron is to find a new coordinate system to

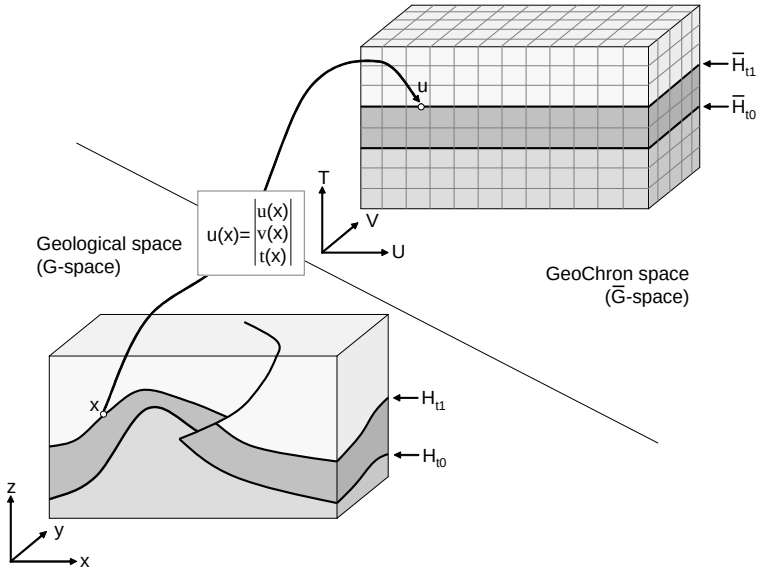


Figure 1.2: This figure shows the geological space (G-space) and the GeoChron space ($\overline{G}$-space). The horizons H_t of the G-space are planes $\overline{H}_t$ in the $\overline{G}$-space. The fault has disappeared. The mapping between a point $\mathbf{x}$ in the G-space and its corresponding location $\mathbf{u}$ in the $\overline{G}$-space is performed by the vectorial function $\mathbf{u}(\mathbf{x})$. $\mathbf{u}(\mathbf{x})$ is also called GeoChron parametrization of the geological space. After [Mallet, 2004].

perform computations on sedimentary models. GeoChron proposes a curvilinear system (u,v,t) for folded geological structures where (u,v) are parallel to the horizons (geographical component) and t is orthogonal to the layers (chronological component). Let us consider a geological model G in present time embedded in the $3d$-Euclidean space (x,y,z). This model was folded and may be intersected by faults. Dissociated parts form so-called fault blocks. Any point $\mathbf{x}$ of G in the Euclidean space can be expressed by a given right handed coordinates system with its orthonormal vectors $(\mathbf{X},\mathbf{Y},\mathbf{Z})$.

$$\mathbf{x} = x\cdot\mathbf{X} + y\cdot\mathbf{Y} + z\cdot\mathbf{Z} \tag{1.30}$$

To simplify the following notations $\mathbf{x}$ is also used to denote the column matrix $\mathbf{x} = [x,y,z]^t$ containing the components of $\mathbf{x}$ in the orthonormal $\mathbf{X},\mathbf{Y},\mathbf{Z}$ frame. From seismic interpretation and well data a set of horizons $\{H_{t_0}, H_{t_1}, ..., H_{t_n}\}$ can be extracted and sorted chronologically by geological time: $t_0 < t_1 < \cdots < t_n$, using the convention $t_i < t_j \Leftrightarrow H_{t_i}$ is older than H_{t_j} (see figure 1.2). One property of GeoChron is that the geological times $\{t_0, t_1, ..., t_n\}$ need not be known and can be chosen arbitrarily honoring their inequalities. The GeoChron model reconstructs images of sediments at the time of their deposition. As consequence, the GeoChron space is unfolded and unfaulted. Therefore a new orthonormal system defined by the vectors $\mathbf{U},\mathbf{V},\mathbf{T}$ is introduced where $\mathbf{T}$ is orthogonal to the deposited surface and oriented from the center of the earth to its surface. This deposited surface $\overline{H}_t$ of geological time t is considered to be a plane parallel to the vectors $(\mathbf{U},\mathbf{V})$. As a consequence, any given reference point $\mathbf{p}_0$ of $\overline{H}_t$ can be used to define a paleo-geographic coordinate system (u,v) defined by $(\mathbf{U},\mathbf{V})$.

$$\mathbf{p} \in \overline{H}_t \Leftrightarrow \exists (u,v) \in \mathbb{R}^2 : \mathbf{p} = \mathbf{p}_0 + u\cdot\mathbf{U} + v\cdot\mathbf{V} \tag{1.31}$$

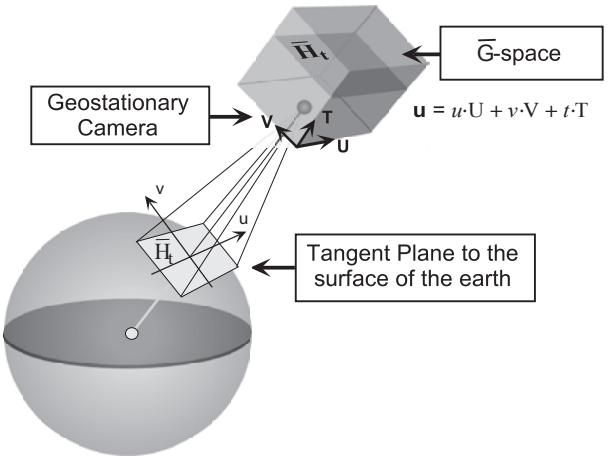


Figure 1.3: This figure describes the GeoChron space as continuous accumulation of images $\overline{H}_t$ of the earth at geological time t taken by a geostationary camera in chronological order. After [Mallet, 2004].

So any particle deposited at geological time t can be located using (u,v). The GeoChron space ($\overline{G}$-space) is a $\mathbf{U},\mathbf{V},\mathbf{T}$ parametric space of a continuum of pictures $\{H_{t_0},H_{t_1},...,H_{t_n}\}$ throughout geological time (see figure 1.3). In this parametric space each image H_t is orthogonal to $\mathbf{T}$ and intersects the geological time scale at t. All paleo-geographic coordinate systems (u,v) of H_t are parallel to $(\mathbf{U},\mathbf{V})$. In this parametric space, each particle deposited at geological time t at the paleo-geographic coordinates (u,v) can be located by:

$$\mathbf{u} = u\cdot\mathbf{U} + v\cdot\mathbf{V} + t\cdot\mathbf{T} \tag{1.32}$$

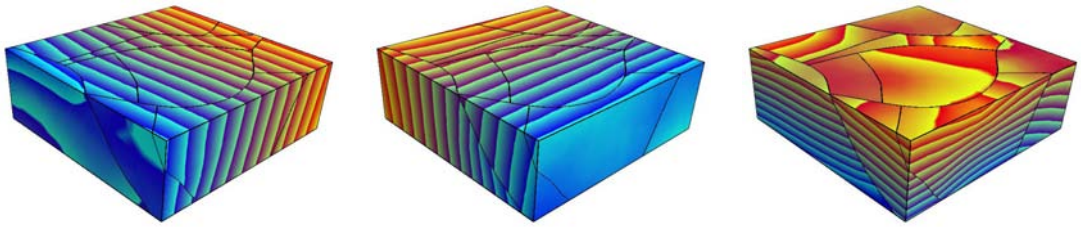
Again, $\mathbf{u}$ also denotes the column matrix containing the components (u,v,t) in the $\mathbf{U},\mathbf{V},\mathbf{T}$ parametric space: $\mathbf{u} = [u,v,t]^t$. To map each point $\mathbf{x}$ of the G-space to a location $(u(\mathbf{x}),v(\mathbf{x}),t(\mathbf{x}))$ in the $\overline{G}$-space, GeoChron defines three functions $u(\mathbf{x})$, $v(\mathbf{x})$ and $t(\mathbf{x})$. These functions can be represented by a vectorial form $\mathbf{u}(\mathbf{x})$ that is called GeoChron parametrization of the geological space:

$$\forall\,\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in G \xrightarrow{u} \mathbf{u}(\mathbf{x}) = \begin{bmatrix} u(\mathbf{x}) \\ v(\mathbf{x}) \\ t(\mathbf{x}) \end{bmatrix} \in \overline{G} \tag{1.33}$$

For each model in the G-space, there exists an infinite number of $\overline{G}$-parametrizations that can be deduced from any existing $\overline{G}$-parametrization by re-scaling the geological time axis t or rotating and translating the paleo-geographic frame (u,v).

$$\mathbf{u}'(\mathbf{x}) = \begin{bmatrix} u'(\mathbf{x}) \\ v'(\mathbf{x}) \\ t'(\mathbf{x}) \end{bmatrix} = \left(\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \cdot \begin{bmatrix} u(\mathbf{x}) \\ v(\mathbf{x}) \end{bmatrix} \right) + \begin{bmatrix} \Delta_u \\ \Delta_v \\ \Delta_t \end{bmatrix} \tag{1.34}$$

$F(t)$ is an arbitrary monotonic increasing function, α an arbitrary rotation angle and Δ_u, Δ_v and Δ_t arbitrary constants. Figure 1.4 shows the three components of the GeoChron parametrization painted with a periodic colormap on a model in the geological space. The fault block boundaries are highlighted with black outlines.



(a) U (b) V (c) T

Figure 1.4: This series of figures shows the three components of the GeoChron parametrization applied to a model in the geological space. The fault block boundaries are highlighted as black outlines. Data by courtesy of Total.

1.3 Summary

This chapter presented definitions and techniques that are used throughout this work. The DSI method is used to compute and modify implicit models defined on tetrahedral meshes (see chapter 5 and 6). For the rapid manipulation of geological horizons a real-time extension of DSI was developed (see chapter 6) to speed up its performance to interactive frame rates. GeoChron is the basis for a number of advanced visualization techniques (see chapter 3) and enables consistent model updates (see chapter 6).

Chapter 2

Iso-Value Surface Interpolation

Contents

2.1 Visualization of Volumetric Data

Nowadays, visualization of volumetric data, is due to the advances in consumer graphics hardware – driven by the multimedia industry – no longer restricted to dedicated graphic workstations. As shown in the introduction, tetrahedral meshes are more and more used for geo-modeling applications. Visualization of tetrahedral meshes in G◯CAD was limited to boundary visualization. Thus, the graphics routines presented here were written from scratch. When dealing with visualization of volumetric data one can distinguish in general three methods:

Slicing: The volume data set is cut with an arbitrarily oriented plane extracting a $2d$-subset of the data set's information.

Iso-value surface: An iso-value surface is built by extracting a level set where a property takes a constant value. This $3d$-surface consists of polygonal patches.

Volume rendering: The whole volume or a sub-volume of the data set is visualized at once by mapping the property values to graphical attributes like color and transparency.

Figure 2.1 illustrates these three methods applied to a bonsai data set defined on a Cartesian grid. The visualization was performed using the G◯CAD-plugin VolumeExplorer [Castanié et al., 2005]. The presented work handles the first two methods – slicing the

(a) Cross-section (b) Iso-value surface (c) Volume rendering

Figure 2.1: These figures illustrate the different visualization techniques (planar section (a), iso-value surface (b) and volume rendering (c)) for volumetric data [Castanié et al., 2005]. The bonsai data set is defined on a Cartesian grid with 8,388,608 cells. Data by courtesy of OpenQVis.

volumetric data by planes or arbitrary bounding boxes, respectively, and extracting iso-value surfaces of scalar properties – for tetrahedral meshes.

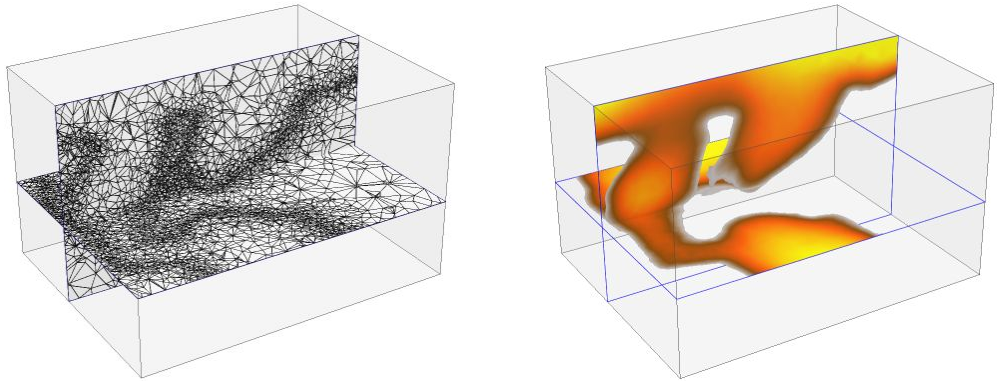
2.1.1 Modeling Rendering Paradigm

The modeling rendering paradigm is widely used in computer games, animation and CAD applications where the calculation or creation of the model can be separated from its actual image [Angel, 2003]. Creating an image is a two-stage process:

Modeling (geometrical primitive extraction) defines a scene by describing its elements. For our applications this is the definition of the geometry of all polygons and line segments of the iso-value surfaces to be extracted from the tetrahedral mesh. The calculation of these geometrical primitives is performed by software running on the CPU (Central Processing Unit).

Rendering uses the model description to build a view of the model using additional information like color or textures. The rendering is performed in the OpenGL graphics pipeline and is processed by the GPU (Graphics Processing Unit).

Both model (geometry) and rendering data (texture coordinates, color indices) are stored independently in so-called vertex arrays. A vertex array is linearly allocated system memory (RAM) or dedicated memory of the graphics hardware (Video RAM). The interface between the modeling and rendering process is just the hardware address of the vertex array. Compared with display lists, vertex arrays allow a clearer distinction between the model and the view and can be manipulated at any time. Figure 2.2 shows a tetrahedral model of a salt dome. Figure 2.2(a) illustrates the wire frame of the model and figure 2.2(b) shows the same model rendered with a pseudo distance function to the salt dome. The penetration of the salt dome with the cross-sections is clipped. More information on this salt dome model can be found in chapter 5.



(a) Model of two cross-sections (b) View of two cross-sections

Figure 2.2: (a) illustrates the model of two cross-sections through a tetrahedral mesh containing a salt dome. The geometrical primitives are visualized by a wire frame. (b) illustrates the same model rendered with a pseudo distance function to the salt dome surface. The penetration of the salt dome with the cross-sections is clipped. Data by courtesy of Shell.

2.1.2 A Very Short Trip Down the OpenGL Graphics Pipeline

Figure 2.3 shows a schematic view of a programmable OpenGL graphics pipeline. The OpenGL commands enter the pipeline from the left side. These commands define both the model and the view. The result of the graphic pipeline is written into the frame buffer.

Evaluator The Evaluator stage can be used to approximate curves or surfaces by computing polynomial functions over the input data. This feature is not used by our implementation and is not further described.

Vertex unit The vertex unit operates on geometrical primitives like points, line segments or polygons. In this unit the vertices are transformed, then lighting operations are performed if requested. A user-defined vertex program can be used to add custom functionality to this stage. Finally, the primitives are clipped to the viewing volume.

Rasterizer The rasterizer generates a two-dimensional, discrete representation of the points, line segments and polygons. The result is a set of fragments and for each fragment a frame buffer address, a color value and texture addresses are computed.

Fragment unit The fragment unit operates on every single fragment generated by the rasterizer. In this stage the fragment value (color and opacity) can be altered by blending, conditional and logical operations and many more. Again this stage can be customized by a user-defined fragment shader. The result of the fragment unit is stored in the frame buffer.

This work makes use of vertex shaders to generate and transform texture coordinates for multitexturing applications. Adapted blending functions for geological applications can be implemented by fragment shaders. These shader programs are written using the CG shading language. A detailed description of CG can be found in [Fernando and Kilgard, 2003; Mark et al., 2003].

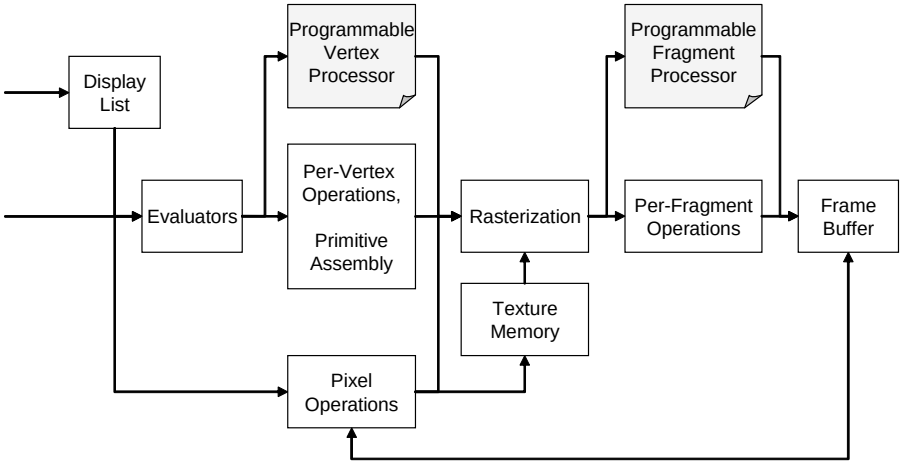


Figure 2.3: This figure illustrates a programmable OpenGL graphics pipeline. OpenGL commands enter the pipeline from the left side. These commands describe both the model and the parameters needed to create the view. The vertex and fragment unit can be customized by user-defined programs (vertex and fragment shaders). The result is written into the frame buffer. After [Segal and Akeley, 2003].

2.2 Review of Iso-Value Surface Extraction Methods

Iso-value surface extraction consists in general of two main parts:

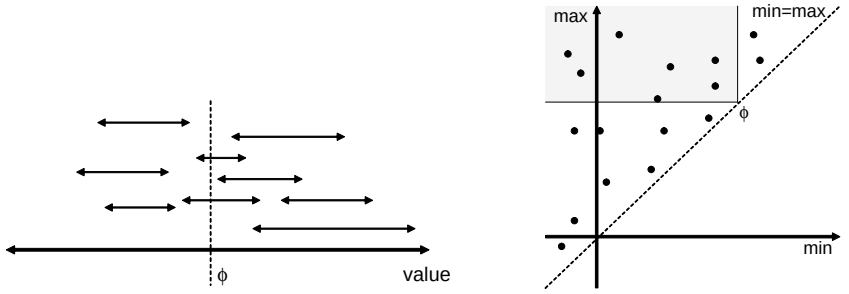
Computation of intersected cells The *brute force* method evaluates for each iso-value surface all tetrahedrons of a complex. Cignoni et al. [1997] postulate that on average only $\mathcal{O}(n^{(d-1)/d})$ cells are intersected by an iso-contour in the d-dimensional space. This process rapidly becomes the bottleneck of the overall process. A computation of the subset of cells that are cut by an iso-value surface and performing the remaining operations on this subset speeds up the overall process. Therefore, two main methods can be identified: *partitioning* and *propagation*.

Interpolation of intersection polygon Classic approaches perform the intersection calculation on the CPU. The introduction of programmable graphics hardware allows polygonization to be performed directly on the GPU.

In the following, the most important methods for cell classification and interpolation are presented.

2.2.1 Partitioning

Partitioning methods treat cells independently from each other. Hereby, the tetrahedrons are classified to speed up the access to cells that are affected by an iso-value surface.



(a) $1d$-value space (b) $2d$-span space

Figure 2.4: (a) illustrates the $1d$-value space. In this space, every tetrahedron is defined by an interval spanning its property range. (b) shows the $2d$-span space, where a tetrahedron is defined as point. The coordinates are the property extrema of the tetrahedron.

Geometric Space Decomposition

An octree is a classic data-structure for spatial partitioning. Plate et al. [2002] use this technique to segment very large structured seismic data in order to create planar slices. To access iso-value surfaces via an octree, Wilhelms and Van Gelder [1990] assign to each octree node the local property extrema of cells covered by this node. Later, only the octree nodes that span the iso-value are used for computation. It is obvious that only the extraction of iso-value surfaces with a high grade of geometrical coherence are significantly accelerated.

Value Space Decomposition

Scalar properties with a high geometric frequency can be better processed in the value space, in which a tetrahedron T can be described by a single range R.

$$R(T) = [\min\varphi(\alpha_i), \max\varphi(\alpha_i)] \quad \forall\, \alpha_i \in T \tag{2.1}$$

According to this description, each tetrahedron covers an interval in the $1d$-value space and is a point in the $2d$-span space. In the $1d$-value space, all tetrahedrons are intersected by an iso-value surface whose interval contains the iso-value ϕ. In the $2d$-span space, all tetrahedrons that are covered by the area defined by the iso-value are intersected (see figure 2.4). Bucket methods subdivide the property range into equally spaced or adaptive intervals. Each bucket stores references to the cells that intersect the bucket interval. Gallagher [1991] proposed the Span Filter method to diminish the memory effort of bucket methods but reduced the lookup performance at the same time. Giles and Haimes [1990] used two sorted cell lists ordered by the minimum and maximum of a cell value. Then, the maximum interval $\Delta\varphi$ of any cell is determined. The set of active cells (all cells intersected by the iso-value surface) is built in two steps. First, the active cells $\mathscr{C}$ are initialized with all cells starting from the iso-value $\phi - \Delta\varphi$ to ϕ from the minimum list. In a second step, all cells from $\mathscr{C}$ whose maximum value is less than ϕ are removed from $\mathscr{C}$. Shen and Johnson [1995] extended the algorithm of Giles and Haimes [1990]. Their Sweeping

Simplices method links the entries of the minimum list with a pointer to the corresponding entries to the maximum list. For an iso-value ϕ the cells in the minimum list with a value less than ϕ are determined and tracked down in the maximum list. The tracked cells in the maximum list are marked with a flag. As soon as the value of the cell in the maximum list is less than ϕ, the process is stopped. All flagged cells in the maximum list are intersected by the iso-value surface.

Trees are also widely used in value space decompositions. An interval tree is a binary search tree where all nodes define a split value [Edelsbrunner, 1980; McCreight, 1980]. When a cell is inserted into the tree it moves down the left branch if its interval lies beneath the split value of the node or the right branch if its interval lies above the split value. If the interval spans the split value for the current node the traversal is stopped and the interval is stored at that node. At each node, the cell intervals are stored in a minimum and maximum sorted list. The storage complexity of an interval tree is $\mathcal{O}(n)$. A disadvantage of the interval tree is that its node lists cannot be arranged arbitrarily. Segment trees do not have this limitation. An interval $[a,b]$ with $a,b \in 1,...,n$ is decomposed into elementary segments $[i,i+1]$, $1 \leq i < n$. The leaves store the elementary segments and the nodes sum up the underlying segment ranges. The storage complexity of an interval tree is $\mathcal{O}(n \log h)$, where h is the height of the segment tree. Both for interval trees and segment trees the query complexity is $\mathcal{O}(k + \log n)$, where k is the size of the response. The NOISE algorithm by Livnat et al. [1996] classifies the cells into a Kd-tree [Bentley, 1975]. Kd-trees are multidimensional binary search trees. The root node stores the median of the property. The cells are partitioned according the median and stored recursively in the two sub trees. Populating the Kd-tree can be performed in $\mathcal{O}(n \log n)$ and a query to such a tree performs in the worst case with a complexity of $\mathcal{O}(k + \sqrt{n})$ with k the number of the output cells.

The value space methods are among the most popular algorithms for visualization. The method developed in the context of this work is also a value space method that computes an octree decomposition in a parametric space.

2.2.2 Propagation

Contour propagation heavily uses topological information of the cells. For a continuous scalar field, an iso-value surface that cuts a cell also cuts the adjacent cell sharing the intersected cell face. This information can be used to accelerate the intersection compared to methods that access the cells in a random order. The propagation is initiated on a seed cell of the iso-contour and traversed over the volume until the original seed cell is traversed. Propagation methods mainly differ in the way the seed cells are computed. An overview over different propagation methods can be found in [Bajaj, 1999]. Due to the high preprocessing costs and the difficulty of parallelization, propagation methods are not further pursued.

2.2.3 GPU Based Methods

Westermann and Ertl [1998] introduced a method to extract shaded iso-value surfaces based on volume rendering using $3d$-textures. The iso-value surface is directly rendered without computing a polygonization. Röttger et al. [2000] extended this method by a hardware accelerated marching cells algorithm based on projected tetrahedrons (PT) [Shirley and Tuchman, 1990]. A disadvantage of the PT algorithm is the cost-intensive visibility ordering of the cells. After the visibility ordering the tetrahedrons are classified according to their projection and decomposed into triangles. Color and opacity are determined by ray integration.

The programmability of modern GPUs invites to perform iso-value surface interpolation directly on the graphics hardware. Regarding a tetrahedral mesh as stream of independent simplices a SIMD (Single Instruction Multiple Data) architecture seems to be perfect for such calculations. Pascucci [2004] introduced a method to perform iso-value surface interpolation on the vertex unit of a GPU. Therefore, all tetrahedrons are passed as quadrilaterals to the GPU. Property values are stored in the attribute registers of a vertex. The intersection is computed by lookup tables that store the configurations of the Marching Tetrahedrons algorithm in the constant registers. Cells that are not intersected by the iso-surface define a quadrilateral with four coincident points and are discarded by the rasterizer. An extension of this method was developed by Buatois et al. [2006]. Geometry, property and lookup tables are stored as texture maps. Klein et al. [2004] proposed a method for iso-value surface extraction of unstructured grids on the fragment unit of a GPU. The cell geometry is coded into texture maps and the iso-value surface is rendered directly into one quadrilateral. Of course, the GPU-based interpolation methods can be combined with partitioning techniques to gain further speedup. These methods only develop their performance when all data are stored in the video RAM. Geological co-rendering applications occupy video RAM with large $3d$-texture maps. Further, topological information is hard to access and fragment and texture units may already be used by other applications. For these reasons this approach was not further pursued. After this review of iso-value extraction methods the algorithms developed in this work are introduced.

2.3 Iso-Value Surface Extraction

As already mentioned, this work deals with the rendering of planar sections and the extraction of iso-value surfaces. The computation of planar slices can be regarded as a special case of an iso-value surface. So, the next sections only reflect iso-value surfaces. The iso-value surface extraction methods – developed in this work – follow several guidelines, such as:

- To develop a generic framework for future extensions.
- To keep accessibility to topological information of the tetrahedral mesh.

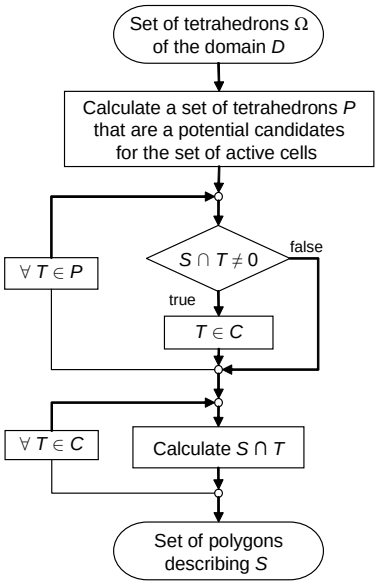


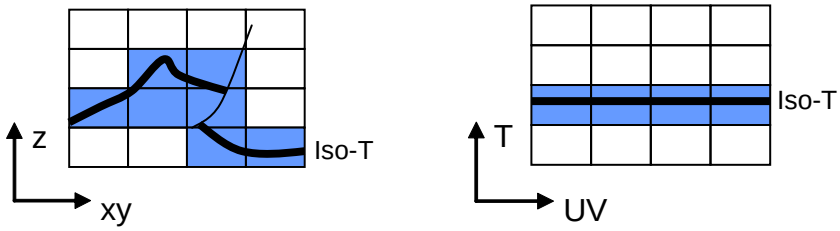
Figure 2.5: This flowchart shows the workflow for the iso-value surface intersection. In the first step a subset $\mathscr{P} \subset \Omega$ is computed. This set contains possible candidates of cells to be intersected by $\mathscr{S}$. The second step computes the set of active cells $\mathscr{C} \subset \mathscr{P}$ with $\forall\ T \in \mathscr{C} : T \cap \mathscr{S} \neq \emptyset$. Finally the polygonization of $\mathscr{S}$ is computed.

- To use explicit polygonization for applications like surface reconstruction or geophysical applications (e.g. ray tracing).

Figure 2.5 shows a flowchart of the iso-value surface extraction method. The set of n tetrahedrons $\Omega = \{1,2,...,n\}$ covering a domain $\mathscr{D}$ in $\mathbb{R}^3$ are considered as independent streams of cells, whereas the topological information is accessible. One tetrahedron T is defined by its four vertices in $T = T(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ where $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$ are embedded into the $3d$-Euclidean space. The values $\{\varphi(\alpha_0), \varphi(\alpha_1), \varphi(\alpha_2), \varphi(\alpha_3)\}$ of the property $\varphi(x,y,z)$ are attached to the vertices of a tetrahedron T. In a first step, a subset $\mathscr{P} \subset \Omega$ of potential candidates to be intersected by an iso-value surface $\mathscr{S}$ is computed. The elements of $\mathscr{P}$ that pass an intersection test form the set of active cells $\mathscr{C} \subset \mathscr{P}$ with $\{T \in \mathscr{C}\} \Leftrightarrow \{\mathscr{S} \cap T \neq 0\}$. In a final step, the polygonization of $\mathscr{S}$ is computed on $\mathscr{C}$. The different phases of this workflow are described in more detail in the following sections.

2.3.1 Partitioning – Parametric Interval Octree

Geometric partition of the $3d$-Euclidean space only improves performance for geometric intersections (e.g. planar slices). The method proposed by Wilhelms and Van Gelder [1990] requests a geometrically coherent iso-value surface to perform adequately. The basic idea behind the octree decomposition developed in this work is to create the octree in a parametric space. In this space, the octree axes are not aligned with the geometric coordinates but with the coordinates of a multi-dimensional value space. In this space, iso-value surfaces are planes. The intersection with the octree can be performed as a



(a) Geometric space (b) GeoChronic space

Figure 2.6: These figures show decomposition methods for the iso-value surface extractions. (a) shows a regular decomposition in the geometric space, where the iso-contour spans seven cells. In the parametric space (b) the same iso-value surface is a plane and covers a minimum number of cells.

geometric intersection and the octree performs in an optimum way. The default parametric space used for decomposition is the GeoChron parametrization (see section 1.2). But any other alignment of the octree axes can be considered, according to the application. Complete octrees with a height $h < 6$ are used and the population with the cells is derived from the method used for interval trees. Thus, both storage as well as preprocessing complexity are linear $\mathcal{O}(n)$ for the acceleration of up to three properties at the same time. An intersection operation with an iso-value ϕ of $\varphi(x,y,z)$ is a geometric intersection with a plane orthogonal to the axis of this property in the value space. The complexity for computing the set $\mathscr{P}$ is $\mathcal{O}(\log h)$. Each tetrahedron $T \in \mathscr{P}$ has to be tested for intersection with $\mathscr{S}$. For all tetrahedrons $T \in \mathscr{P}$, the signed distance of the property $\varphi(x,y,z)$ to the value ϕ of the iso-value surface $\mathscr{S}_\phi$ is computed at the four vertices $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$ of T. If the arithmetic sign of the distance of any of the four vertices is different to the others or the distance is zero then $T \in \mathscr{C}$. The computation of the signed distance is only performed once for each node in the tetrahedral mesh. The overall complexity of computing the active cells using octree decomposition in the value space is $\mathcal{O}(k + \log h)$ with $k = |\mathscr{P}|$. Once the active cells are computed, the intersection calculations have to be performed for each $T \in \mathscr{C}$.

2.3.2 Marching Tetrahedrons

The Marching Cubes [Lorensen and Cline, 1987] algorithm is a well-known method to extract triangulated iso-value surfaces from $3d$-data. The Marching Tetrahedrons [Cignoni et al., 1998] approach is a specialization of this method which is based on a tetrahedral decomposition. Compared to the Marching Cubes the Marching Tetrahedrons does not have to deal with ambiguous iso-value surface trends. Every edge of T can be intersected by $\mathscr{S}$, or not. This leads to 2^6 cases. The trivial case where $\mathscr{S} \cap T = 0$ (see figure 2.7(a)) was already excluded computing the set of active cells $\mathscr{C}$. Disregarding the permutation of the edges and taking symmetry into account, these cases can be reduced to five topological different possibilities where $\mathscr{S} \cap T \neq 0$ (see figure 2.7(b) to 2.7(f)). The cases where the iso-value surface is coincident with a node, edge or face of the tetrahedron (see figure 2.7(d) to 2.7(f)) are deduced from the cases of figure 2.7(b) and 2.7(c). When three edges of T are intersected by $\mathscr{S}$ then $\mathscr{S} \cap T$ is a triangle. If four edges of T are intersected

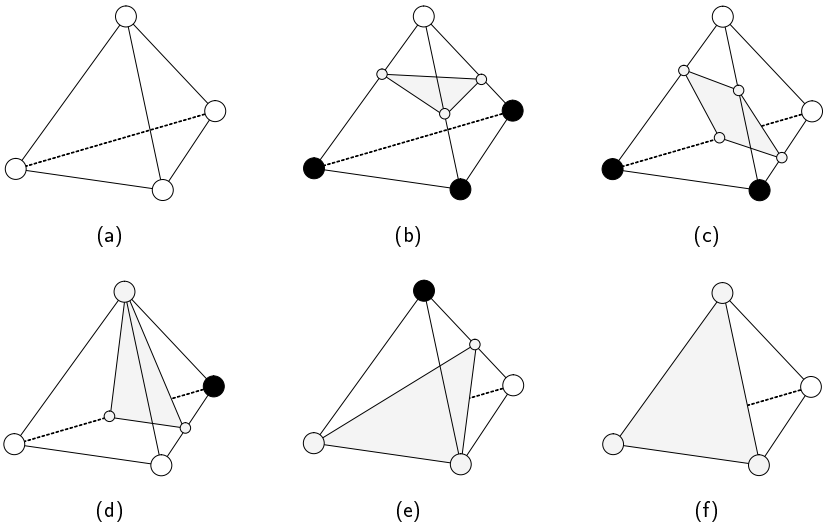


Figure 2.7: These figures show the topological cases of the Marching Tetrahedrons method. (a) shows the trivial case where $T \cap \mathscr{S} = \emptyset$. If $\mathscr{S}$ intersects T on three edges the intersection polygon is a triangle (b) and if $\mathscr{S}$ intersects T on four edges the intersection polygon is a quadrilateral (c). Cases where $\mathscr{S}$ is coincident with a vertex, edge or face of T (d)-(f), can be deduced from the cases (b) and (c).

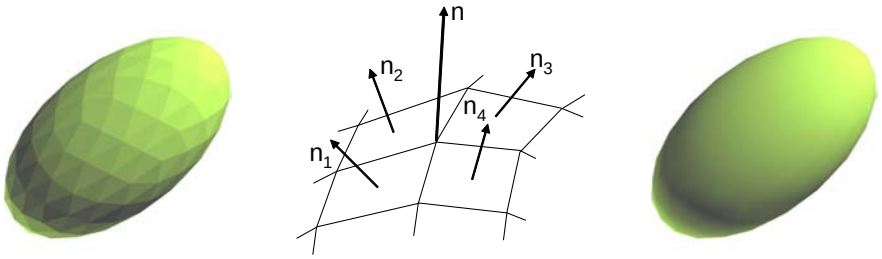
by $\mathscr{S}$ then $\mathscr{S} \cap T$ is a quadrilateral. This is the precise solution if $\mathscr{S}$ is locally planar and an adequate approximation for non planar surfaces. The intersection computation is performed by interpolating the iso-value ϕ of $\mathscr{S}$ for each intersected edge of T. Linear interpolation is the simplest and fastest method to extract the intersection polygon of $\mathscr{S}_\phi$ with T. An edge of a tetrahedron is intersected by the iso-value surface if the difference $\Delta_0 = \phi - \varphi(\alpha_0)$ at point α_0 and the difference $\Delta_1 = \phi - \varphi(\alpha_1)$ at point α_1 have a different sign. The point of intersection s between an edge of the tetrahedron and the iso-value surface is computed with formula (2.2).

$$s = \alpha_0 + \frac{\Delta_0}{\Delta_0 - \Delta_1} \cdot (\alpha_1 - \alpha_0) \tag{2.2}$$

Piecewise smooth cubic Hermite interpolation was evaluated to obtain smoother iso-value surfaces. In practice, the improvement gained by higher order interpolation is very small and generally does not justify the much higher computational effort [Frank and Mallet, 2004]. More information on numerical properties of tetrahedrons cut by an iso-value surface can be found in Royer [2005].

2.3.3 Lighting Normals

For every lighting operation in computer graphics the knowledge of the normal vectors at the vertices is essential. If the normal vector of a polygon is applied to every vertex of the facet, the graphic result is a faceted surface as shown in figure 2.8(a). To increase the smoothness of the lightened surface, it is necessary to average the lighting normal of adjoining polygons at a vertex (e.g as illustrated in figure 2.8(b)). The vectors $\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3, \mathbf{n}_4$ are summed up and normalized afterwards. The resulting lighting vector is attached to the



(a) Flat shading (b) Interpolated normals (c) Gouraud shading

Figure 2.8: (a) shows a flat-shaded ellipsoid. The interpolation of the normals of the facets on shared vertices (b) leads to a much smoother shading of the ellipsoid (c).

shared vertex. Proceeding in this way for every point of the polygons leads to a smooth surface as shown in figure 2.8(c). Any normal of $\mathscr{S}$ can be described by the gradient vector $\nabla\varphi$ of $\varphi(x,y,z)$. Therefore $\nabla\varphi$ is computed as a preprocessing step for all nodes of the tetrahedral mesh (see section 1.1.1). The gradient at an intersection of $\mathscr{S}$ and T is obtained by interpolation. Using $\nabla\varphi$ as lighting vector allows to treat the polygons independently from each other as the notion of shared vertices is not needed. This is a major step towards parallelization (see section 2.3.6).

2.3.4 Generic Implementation

The C++ programming language [Stroustrup, 2000] offers a wide range of possibilities to realize almost every design pattern ranging from pure procedural to object or aspect-oriented and generic programming.

Generic Design Patterns

Generic programming is widely used to implement type-independent container classes like the C++ Standard Template Library (STL) [Stepanov and Lee, 1994]. Besides for implementing container classes, generic programming can also be used to compete with well-known object-oriented design patterns like inheritance and polymorphism. C++ realizes the generic programming paradigm by the means of templates. Often C++ templates are linked with code bloat but the new compiler generations which fulfill the ANSI C++ standard only generate binary code for the generic elements which are really used. One disadvantage of using generic programming compared to object-oriented programming is the higher complexity of the code but in return one can gain a lot of opportunities. Object-oriented patterns lead to class hierarchies which imply that a derived class `B` is of the same kind as its base class `A`. This situation often leads to redundant code. But the main disadvantage of such a pattern is the so-called operation polymorphism that comes along with abstract and virtual member methods. Virtual methods are expensive because of their late binding and the involved virtual pointer table lookup. When one uses them in intensive computing like scientific visualization the performance can crash. Unlike virtual meth-

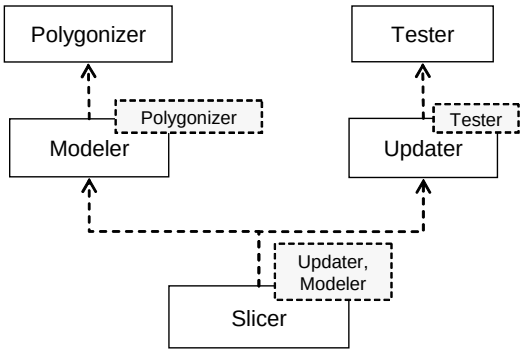


Figure 2.9: This UML class diagram shows the generic composition of the `Slicer` class which manages the computation of an iso-value surface. The `Updater` class computes the set of active cells $\mathscr{C}$ and is parameterized by a `Tester` class. The `Modeler` computes the polygonization of $\mathscr{C}$. Intersection and interpolation algorithms are defined by the `Polygonizer` class.

ods, the deployment of templates results in a binary code that reaches the performance of dedicated code [Géraud and Duret-Lutz, 2000].

Generic Design Patterns in Scientific Computing

Particularly because of their performance, templates are very popular for the implementation of scientific tasks. For example, Siek and Lumsdaine [1998] developed a generic library based on C++ templates for the high performance computation for numerical linear algebra. The implementation of the CIEL data structure [Lévy et al., 2001] used for the visualization of unstructured grids is based on generic patterns. Remy et al. [2004] introduced GsTL for the generic implementation of geo-statistical algorithms. As shown by the above samples, using generic design patterns is the first choice to solve scientific problems with abstract software components.

Generic Graphic Kernel

The workflow presented in figure 2.5 was decomposed into generic software components. A UML (Unified Modeling Language) class diagram of the graphic kernel is shown in figure 2.9. The parametrized `Slicer` class manages the computation of the model of iso-value surfaces. This class has two generic slots. One for an `Updater` class to compute the set of active cells $\mathscr{C}$ which itself bears a generic parameter for a `Tester` class which extracts the tetrahedrons $T \in \mathscr{C}$ from $\mathscr{P}$. The second generic parameter of the `Slicer` class is the `Modeler` class which contains everything from calculating the intersection $S \cap T \ \forall \ T \in \mathscr{C}$ to managing the storage of the model of $\mathscr{S}$ inside vertex arrays. Therefore the `Modeler` class has a generic parameter `Polygonizer` that performs the intersection operation. The integration of a new algorithm for one of the above-mentioned tasks just implies the development of an appropriate (parametric) class and instancing an object of the `Slicer` class with the new component. This high performance approach grants future maintainability and extensibility.

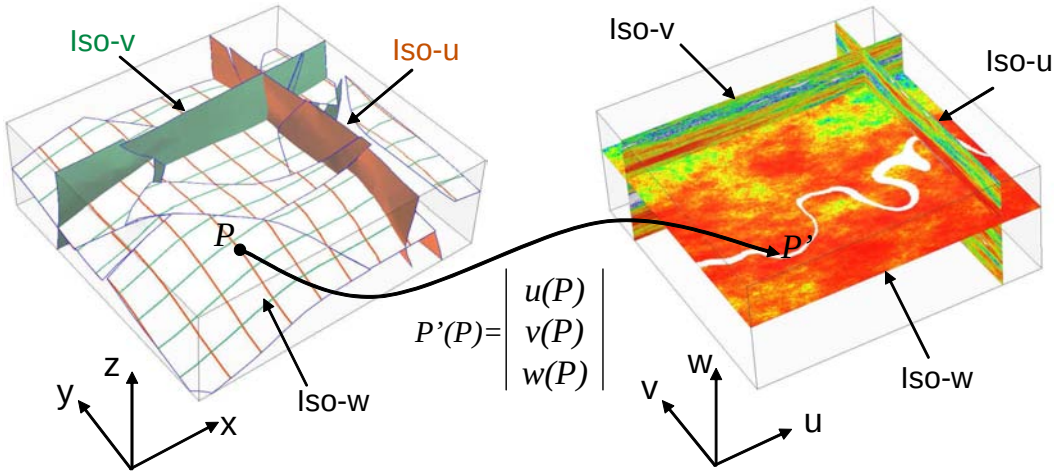
2.3.5 Texture Mapping

So far, this work explained the polygonization of illuminated iso-value surfaces. This section introduces the rendering of these polygonal surfaces with additional information like property values or $3d$-geological information by the means of textures. Textures are one, two, three or four-dimensional patterns. Their primitive elements are called texels. Texture mapping paints the pixels (screen coordinates) of polygons with the content of the texels. Therefore, for each point P of a polygon (world coordinate) a corresponding point P' in the texture map is defined. In fact, texture coordinates are defined per vertex and are interpolated by the rasterizer for every fragment of the polygon. Considering a scalar property, the property values attached to the vertices of a polygon can be used to identify a position in a $1d$-texture map, which is a color-map. The color-gradient of this texture map acts as transfer function. Measurement data are often scattered over the $3d$-geometrical space (e.g. seismic surveys). To map this kind of data, the world coordinates of a polygon can be directly mapped onto the data in the Euclidean space. Simulations are often performed in a so-called computational (parametric) space. To paint a polygon with this kind of data, the parametric coordinates have to be known. Figure 2.10(a) shows the mapping between the world coordinates of a geological model and the texture/parametric coordinate system of the texture map and computational space, respectively. The result is shown in figure 2.10(b). Further, texture mapping enables the rendering of an attribute with a higher frequency than the geometric resolution of the polygon (see figure 2.10(c)).

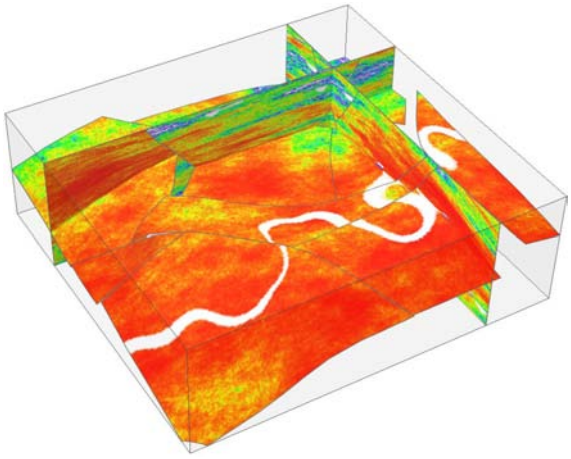
2.3.6 Parallelization

Spatial partitioning into independent subsets of tetrahedrons, as introduced earlier in this chapter, is a first step towards parallelization. Moore's law [Moore, 1965] postulates an exponential growth for the complexity of integrated circuits like CPUs. This exponential growth leads to a doubling of the number of transistors of a CPU every 18 months. Recently, this growth was represented by an increasing clock-rate of the CPU, which is closely linked to the length of its arithmetic pipelines and thermal power problems. The longer a pipeline is, the higher the grade of its instruction parallelism. With increasing pipeline length, the danger of wait conditions increases so that it can be predicted that the limits of this technology will be reached soon. Modern processor architectures follow another approach: parallelism introduced by multi-cores. Each CPU die carries several processor cores. A further step is the assembly of several processors to a symmetric multiprocessing system (SMP). Both multi-cores as well as SMP systems require a parallelized implementation of the algorithms to fully benefit of their performance. One can distinguish two threading models:

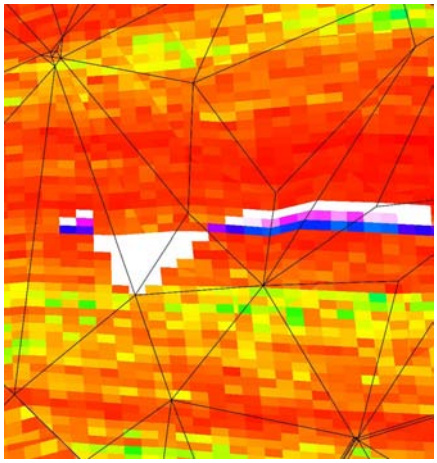
Data parallel threading model works on data parallelism. This means that the same operation is performed on a large amount of independent data. `For`-loops are a typical construct of data parallelism. OpenMP [Mattson, 2000] is an application programming interface that realizes parallel programming for shared-memory machines without the need of manual decomposition of the problem into threads. OpenMP uses a Fork-Join threading model for loops where no dependencies be-



(a) Shows the mapping between the world coordinates (left) and the parametric coordinates (right) of a texture map.

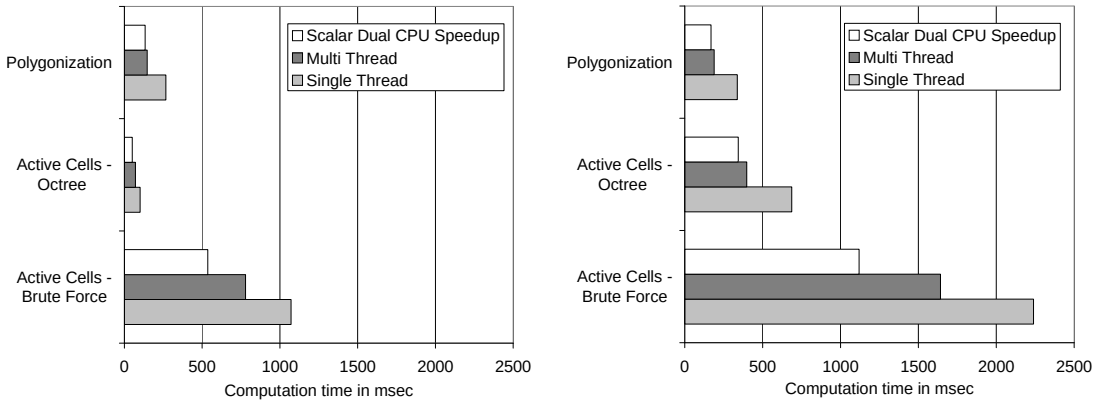


(b) Model defined in world coordinates painted with a *3d*-texture map in a parametric coordinate system.



(c) Magnification of a cross-section. By the means of texturing, attributes with a higher frequency than the geometrical resolution of the polygons can be painted. Polygon borders are outlined in black.

Figure 2.10: Texture mapping with coordinate transformation. Texture coordinates are defined for each vertex of a polygon and are interpolated for every fragment of the polygon. Data by courtesy of Total.



(a) Planar section with 30,980 polygons

(b) Iso-value surface with 29,491 polygons

Figure 2.11: Parallelization benchmark results for a planar section (a) and an iso-value surface (b) extracted from a solid with 848,656 tetrahedrons. The diagrams show the theoretical scalar speedup for a dual SMP machine, the single and multi-threaded timings for the computation of $\mathscr{C}$ using brute-force or spatial partitioning and the polygonization of $\mathscr{C}$.

tween the loop cycles exists. Streaming data or matrix operations are candidates for a compiler based threading model.

Task parallel threading model is a functional decomposition. This model identifies independent tasks of a software module and implements them in different threads. Task parallel decomposition cannot be performed by the compiler. Web and application-servers are classic examples of this kind of decomposition.

Profiling the developed code for the iso-value surface interpolation showed that the majority of CPU-time was spent for the calculation of the set of active cells $\mathscr{C}$ and the polygonization of $\mathscr{C}$. Octree look-ups and rendering of the model took less than 5% of the CPU time. So the computation of $\mathscr{C}$ and its polygonization are good candidates for parallelization. Bartz and Straßer [1999] parallelized the octree decomposition method of Wilhelms and Van Gelder [1990] for the Marching Cubes algorithm. Their main focus lied on the parallel construction of recursive tree hierarchies. The introduced method considers the tetrahedrons to be independent from each other. So the set of possible candidates $\mathscr{P}$ obtained by an octree lookup can be decomposed into subsets. The subsets are processed by parallel threads to compute $\mathscr{C}$. The same method was used to compute the polygonization of $\mathscr{C}$. The parallelization was performed using OpenMP together with Intel's C++ compiler. Certain care was necessary to design the generic classes (see figure 2.9) in a thread-safe way. The results of the parallelization are illustrated in figure 2.11. A solid with 848,656 tetrahedrons was intersected by a plane and an iso-value surface[3]. Timings for the computation of $\mathscr{C}$ with and without partitioning and the polygonization of $\mathscr{C}$ were acquired. The benchmark was performed for a parallelized and a single-thread version of the introduced method. One can clearly see that the polygonization almost

[3]Benchmarks were performed on a dual Intel Xeon SMP system with 2.40 GHz, 2 GByte of RAM and a NVidia Quadro4 900 GLX with 128 MByte memory

reaches scalar speedup. The calculation of $\mathscr{C}$ shows a lower grade of parallelism. This is caused by the synchronized access to the data-structure representing $\mathscr{C}$. In general, planar slicing performs much faster than iso-value interpolation due to the lower complexity of its arithmetic.

2.3.7 Planar Sections

So far, the computation of iso-value surfaces was illustrated. Planar sections orthogonal to the axis of a Cartesian coordinate system can be regarded as special cases of an iso-value surface if the geometry is considered as property. In general, the computation of planar section is much simpler than the iso-value surface interpolation. Further, arbitrarily aligned planes cannot be easily expressed as iso-value surfaces. Parallel to the iso-value surface extraction method, the planar intersection was implemented. The integration to the visualization framework was straightforward, due to its generic design, and will be not further explained. Figure 2.12 shows an application of several planar sections to explore a tetrahedral mesh with a slicing box that can be rotated, scaled and transformed along each axis. Regions outside the box are clipped.

2.4 Summary

This chapter covered a review of methods used for the interpolation of iso-value surfaces. Then, it proposed the developed interpolation method based on the Marching Tetrahedrons method that was accelerated by an octree decomposition performed in a parametric value space. Special care was taken to enable further extensions. This can be easily realized by introducing new parametric classes into the workflow that is based on a generic framework. During his internship at G◯CAD research group, Massenet [2005] developed an extension for visualizing tensor and vector fields defined on tetrahedral meshes using the introduced generic framework (see figure 2.13). Considering the tetrahedral mesh as stream of independent cells, it was possible to implement an efficient parallelization for symmetrical multiprocessor and multi-core machines. The developed iso-value surface extraction method allows to explore geological models interactively in real-time. But, geological models not only consist of properties but also contain complex information like boundaries of geological interfaces, stratigraphy and much more. The following chapter introduces algorithms to visualize different kinds of information at once.

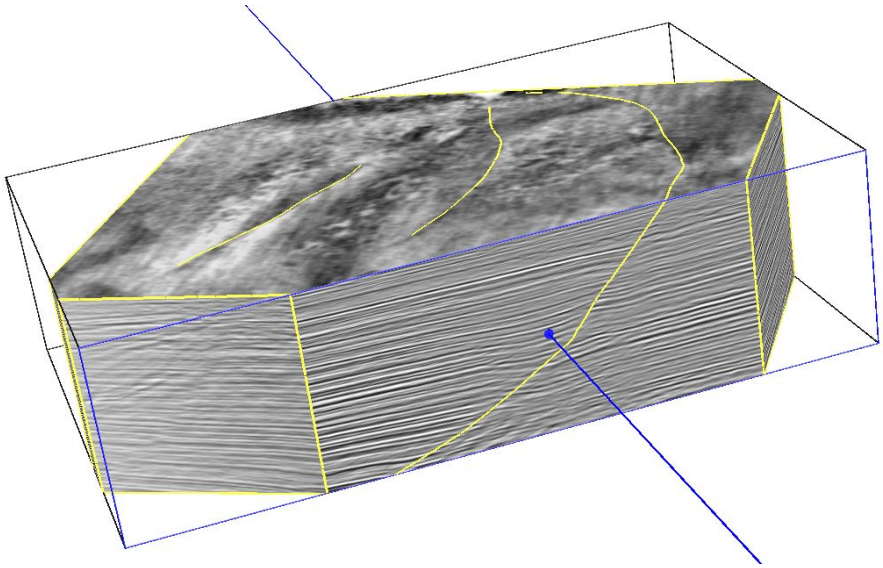
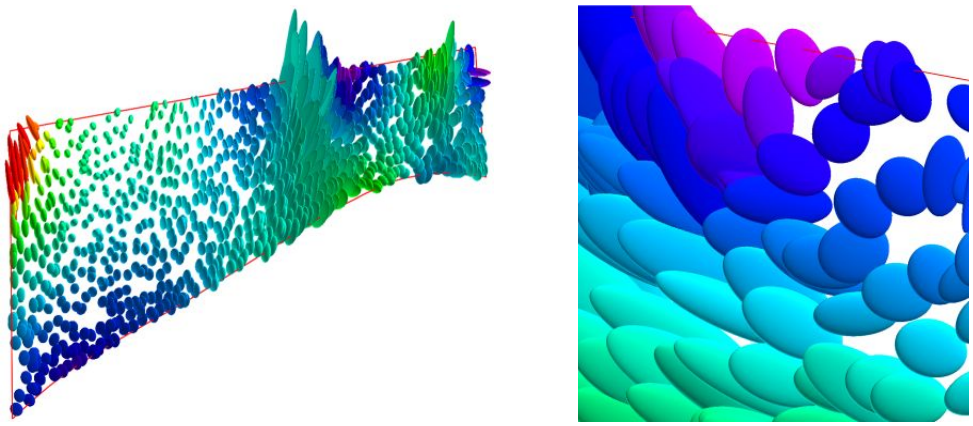


Figure 2.12: This figure shows a tetrahedral mesh explored by a clip box. The visible parts are based on planar sections clipped at the boundaries of the clip box. The model is texture-mapped with seismic data and the faults and volume boundaries are highlighted with yellow outlines. Data by courtesy of Earth Decision.



(a) (b)

Figure 2.13: These figures show the visualization of a tensor field defined on a tetrahedral mesh. The tensorial data are visualized on a cross-section through the simplicial complex. The visualized data are strain tensors used for restoration [Muron, 2005]. The Eigenvectors and Eigenvalues of the tensors are used to define deformation ellipsoids. The ellipsoids are painted with dilatation values [Massenet, 2005].

Chapter 3

Advanced Rendering for Geo-Modeling

Contents

3.1 Introduction

The previous chapter covered the interpolation of iso-value surfaces of scalar properties and arbitrary sections through tetrahedral models. Geological models are rich in information of different nature like topology of fault blocks, stratigraphic layers or distances to geological interfaces like faults or to artifacts like wells. For an optimal information retrieval, the different kinds of information should be combined in a single image as shown in figure 3.1. This figure shows a cross-section and an iso-chron surface (horizon) extracted from a tetrahedral mesh with outlined fault block boundaries. Further, the image shows iso-contours of the GeoChron (see section 1.2) time parametrization and was rendered with a geological property. This chapter introduces new methods to visualize and combine complex geological information on tetrahedral meshes based on the generic implementation of the iso-value surface extraction method.

3.2 Outlines of Faults and Horizons

Faults, fault block boundaries and horizons are very important geological features and their visual representation plays an outstanding role for the understanding of geological models.

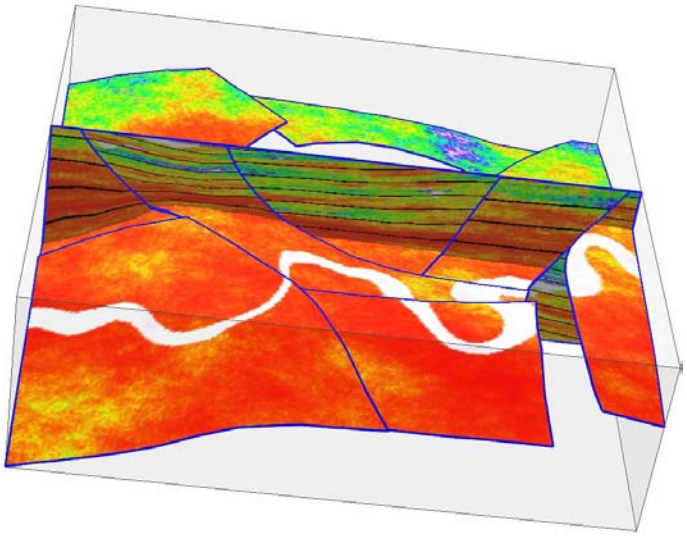


Figure 3.1: This figure shows a cross-section and an iso-T surface extracted from a tetrahedral tessellated model. The faces are texture-mapped with a stochastic channel simulation [Alapetite et al., 2005; Leflon, 2005]. The borders of the fault blocks are outlined in blue. Further, iso-contours of the GeoChron time parametrization are illustrated. Data by courtesy of Total.

3.2.1 Fault Visualization

The basic implementation of the G◯CAD tetrahedral mesh (TSolid), does not include any explicit information about faults or fault blocks. Faults are modeled as micro-topological discontinuities in the graph that represents the tetrahedral mesh (see figure 3.2(b)). The tetrahedral mesh is composed of a set of tetrahedrons described by a boundary-representation (B-Rep) [Weiler, 1988]. Adjacent relationships are stored by so-called Half-Edges and Half-Triangles. These are orientable primitives that are associated to their topological mates. All mates of Half-Edges and Half-Triangles that are dissociated face an discontinuity in the tetrahedral graph.

When a tetrahedron is intersected by a plane this intersection forms a triangle or quad, neglecting the cases where one face, one edge or one vertex of the tetrahedron is coplanar with the intersection plane. Each of these intersections is represented by a data structure called `TetraIntersection (TI)` (see listing 3.1).

Listing 3.1: C++ `TetraIntersection` data structure

```
struct TetraIntersection {
    GLfloat geometry[3];      // vertex coordinates xyz
    GLfloat texture [3];      // 3d−parametric texture coordinate
    GLfloat normal[3];        // gradient vector used for lighting
    GLbool border;            // flag if vertex lies on volume border
    GLbool border_seg [4];    // numbering on which faces of the tetrahedron this
                              // border vertex lies
    GLuint color;             // color index for visualizing scalar properties
};
```

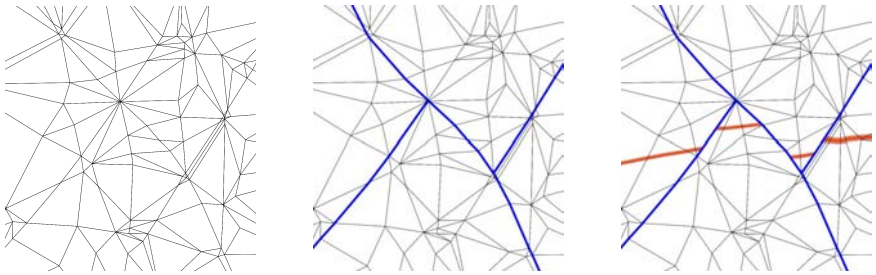


Figure 3.2: These figures show the polygonization of a planar cross-section through a tetrahedral model. (b) illustrates blue outlined faults that are coincident with tetrahedron boundaries. In general, geological interfaces like horizons (red outlines) or salt-bodies are not respected by the topology of the tetrahedral mesh (c).

The `border` variable signals whether this intersection lies on a discontinuity. The boolean array `border_seg` carries one entry for each tetra face. The boolean value `true` signals that the intersection is coplanar with the distinct tetra face and is adjacent to a discontinuity and the value `false` that this is not the case. This initialization can be easily performed by a consecutive iteration and border tests over the faces, edges and vertices of a tetrahedron. The intersection calculation is performed as early evaluation [Meyers, 1997] during the interpolation of the intersection polygon. In the consecutive stage of the modeling algorithm the vertex array data are computed from this information. Then, an iteration over all `TetraIntersection` objects of the intersection polygon is performed to build the vertex tuples that form segments lying on the discontinuity in a first step. In a second step these vertex tuples are used to define line segments in the vertex arrays (see listing 3.2).

Listing 3.2: Pseudo code for building the fault segments from an array of tetra intersections TI

```
if tetra T on border then
  for all i such that 0 ≤ i ≤ 3 do
    count ← 0
    index[4]
    for all j such that 0 ≤ j ≤ 3 do
      if TI[j].border_seg[i] == true then
        index[count] ← i
        count ← count+1
      end if
      j ← j+1
    end for
    for all k such that 0 ≤ k ≤ count−1 do
      draw_line( TI[index[k]].geometry, TI[index[k+1]].geometry )
      k ← k+1
    end for
    i ← i+1
  end for
end if
```

3.2.2 Horizon Visualization

In the previous section, the macro-topological information of unconformities was computed from the micro-topology of the single tetrahedrons. For the visualization of geological horizons, in general, this algorithm is not applicable because geological horizons are not modeled as discontinuities (see figure 3.2(c)). An exception are models built for restoration [Muron, 2005].

In this work, horizons are described as iso-value surfaces of the continuous GeoChron time parametrization. One approach is to perform the intersection calculation between extracted iso-value surfaces and the iso-value surface that represents the horizon explicitly. This procedure is numerically expensive even if spatial partitioning is applied. A more performant solution is the implicit intersection of iso-value surfaces. For example, one can paint a geometrical cross-section with a step function over the GeoChron time. The time of the horizon takes a special value and the intersection of the cross-section with the horizon will be outlined. This algorithm belongs to the multitexturing techniques which are explained in more detail in the following sections.

3.3 Multitexturing Techniques

Recent graphics hardware supports multiple texture maps that can be combined in a single rendering pass. Multitexturing opens the door to a wide variety of applications for geological modeling. Distance maps are a widely used tool to determine the distances to faults or wells inside a $3d$-model. The GeoChron parametrization enables the visualization of stratigraphic columns on unstructured grids. Also complex problems of CSG (Constructive Solid Geometry) like intersection and penetration of arbitrary bodies or surfaces can be solved by texture-based techniques. The following sections introduce the above-mentioned issues and show how they can be combined.

3.3.1 Texture Blending Methods

To co-visualize several attributes, the colors defining these attributes have to be blended for each fragment. This operation is performed on the fragment unit of the OpenGL graphics pipeline (see figure 2.3). The following sections propose two solutions. The first solution uses the built-in OpenGL blending functions together with a multitexture environment. The second solution defines advanced blending functions using fragment shaders.

Multitexturing Using OpenGL Texture Units

The blending of several textures can be performed by applying the texture operations sequentially for each texture on every fragment. An OpenGL texture unit (TU_i) takes the current fragment color C_f and texture lookup color CT_i to compute the fragment output color C'_f. The initial fragment color C_f entering the first texture unit TU_0 is rasterized

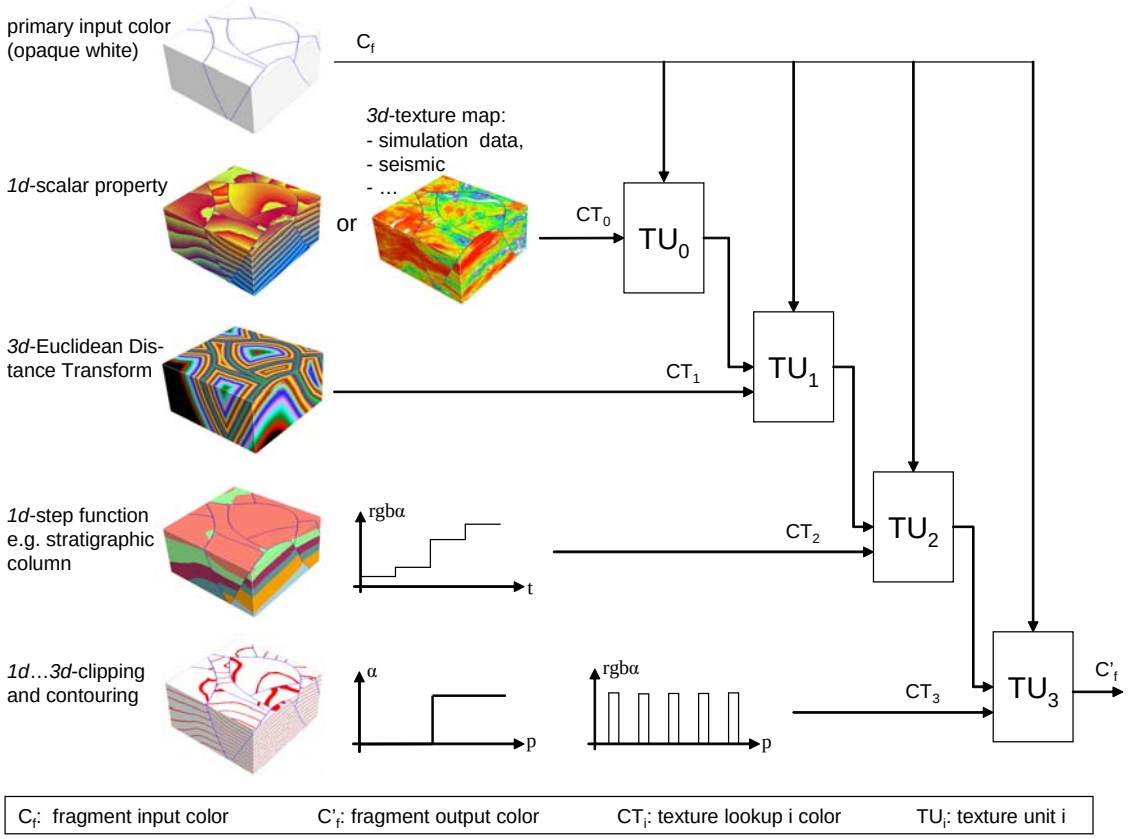
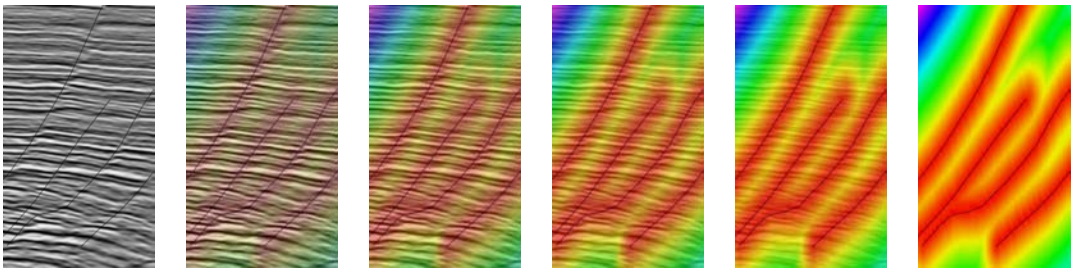


Figure 3.3: This figure illustrates an OpenGL multitexture environment. A texture unit TU_i takes the current fragment color C_f and the texture lookup color CT_i to blend the output color C'_f. Several texture units can be combined in a cascade where the output color of one texture TU_i unit is used as input fragment color of the following texture unit TU_{i+1}.

from an opaque white polygon on which lighting operations were performed. This color is blended with the texture lookup color CT_0 of the texture belonging to TU_0. The result of this blending operation is used as input fragment color for the next texture unit TU_1. This cascade-like multitexture procedure is repeated for all active textures but is limited by the maximum number of texture stages. As texture blending function we use `GL_MODULATE`. The output argument of a texture unit is the product of its input arguments. If the luminance of the multitextured image is too weak, one can scale the output argument with the extension `GL_RGB_SCALE_EXT`. The result is clamped to $[0...1]$ for alpha and each color channel.

The built-in OpenGL texture blending functions are quite limited but useful. For more advanced texture combinations there exists a variety of possibilities. Texture shaders offer a bigger number of built-in texture blending functions and texture combiners join arithmetical operations to create custom texture blending functions [Kilgard, 2004]. One crucial argument not to use these extensions is the lack of support of these extensions by a series of GPUs. Nowadays, programmable graphic pipelines as illustrated in figure 2.3 are common features of recent GPUs. A fragment shader extends the per-fragment oper-



(a) 0.0 (b) 0.2 (c) 0.4 (d) 0.6 (e) 0.8 (f) 1.0

Figure 3.4: This series of figures shows a cutaway of a cross-section through a tetrahedral mesh, co-visualized with seismic data and a distance function to the faults. The color channels of the two attributes where interpolated linearly and the bias (weight of the distance function) of the interpolation is given. Texture blending was performed by the fragment shader of listing 3.3.

ations and allows the definition of customized texture blending functions. This solution is illustrated in the following section.

Customized Blending Functions Using Fragment Shaders

A fragment shader operates on each single fragment created by the rasterizer of the graphics pipeline (see figure 2.3). A fragment shader has access to the data of all active texture maps and the texture coordinates of a fragment, which were interpolated by the rasterizer, are available as parameter. Listing 3.3 shows a basic fragment shader for customized texture blending. This shader uses two $3d$-texture maps and the initial fragment color as input arguments to compute the resulting output fragment color.

Listing 3.3: CG fragment shader for texture blending function

```
void main(float3 tex_coord_0 : TEXCOORD0,
          float3 tex_coord_1 : TEXCOORD1,
          float4 frag_color_in : COLOR,
          out float4 frag_color_out : COLOR,
          uniform sampler3D tex_0,
          uniform sampler3D tex_1,
          uniform float bias )
{
    float4 tex_color_0 = tex3D( tex_0, tex_coord_0 );
    float4 tex_color_1 = tex3D( tex_1, tex_coord_1 );

    frag_color_out = frag_color_in * lerp( tex_color_0, tex_color_1, bias );
}
```

The texture coordinates `tex_coord_0`, `tex_coord_1` and the fragment color `frag_color_in` are passed as values of function parameters. `frag_color_out` is the alias name for the register where the output fragment color is written, `tex_0` and `tex_1` are the two $3d$-texture maps. The CG function `tex3D` is used to perform the texture fragment color lookup. The CG function `lerp` interpolates linearly the two texture lookup

colors `tex_color_0` and `tex_color_1`. The `lerp` function performs a linear interpolation between two vectors like: $c = a * (1 - weight) + b * weight$. The value of weight is passed as a user-defined variable `bias` from the client space. Modifying this variable by the user application results in interactive fading effects between the two texture maps (see figure 3.4). Finally the result of the linear interpolation is multiplied with the input fragment color. The input fragment color is interpolated by the rasterizer from white opaque polygons on which lighting computations where performed. Thus, the lighting is honored by the fragment shader. The fragment shader from listing 3.3 can easily be extended for additional texture maps and blending functions. Of course, color and alpha channels can also be treated independently.

3.3.2 Distance Maps on Unstructured Grids

Distance transforms have been widely investigated for morphological image processing applications. Hereby, image processing is performed on a $2d$ or $3d$-image I consisting of pixels or voxels, respectively. A $3d$-distance map D contains the distance of any given point $p \in I$ to the closest point q of a set of objects O.

$$D(p) = \min \{dist(p,q), q \in O\} \tag{3.1}$$

where $dist(p,q)$ is the Euclidean metric between the point $p \in I$ and $q \in O$. The computation of distance maps is a very important topic for geological applications. An Euclidean Distance Transform (EDT) can be used to represent the distance to geological objects like faults or fractures. Also non-natural objects like wells or the region of influence of model modifications [Tertois et al., 2005] can be considered. Common algorithms for the calculation of n-dimensional distance transforms were developed on Cartesian grids [Cuisenaire, 1999; Meijster, 2004]. These methods depend on equally sized cells with rectangular neighborhood relations. Both assumptions are not fulfilled on an unstructured domain. The explicit application of these algorithms on unstructured grids is consequently not possible. Two methods were elaborated to compute EDT on simplicial complexes. The first method explicitly interpolates a distance function on the nodes of the tetrahedral mesh and the second method uses a distance function defined on a Cartesian grid for texture mapping onto the unstructured domain.

Interpolated Distance Maps

This first method explicitly defines a distance function D on the nodes of the tetrahedral mesh. The function is computed using the Discrete Smooth Interpolation (DSI) algorithm (see section 1.1.2). Points of the object O are used as property control points with a scalar value 0. If O consists of surfaces, unit vectors normal to that faces are installed in the directions the distance is to be computed. These vectors are used to define the direction and the magnitude of the gradient of D. Figure 3.5(a) shows an unconformity together with a set of DSI constraints. The vertices of the tetrahedrons faces that are coplanar with this unconformity are used to define points with a distance zero. Normal unit vectors pointing to the interior of each sub-volume are installed to define the gradient of D. Applying this

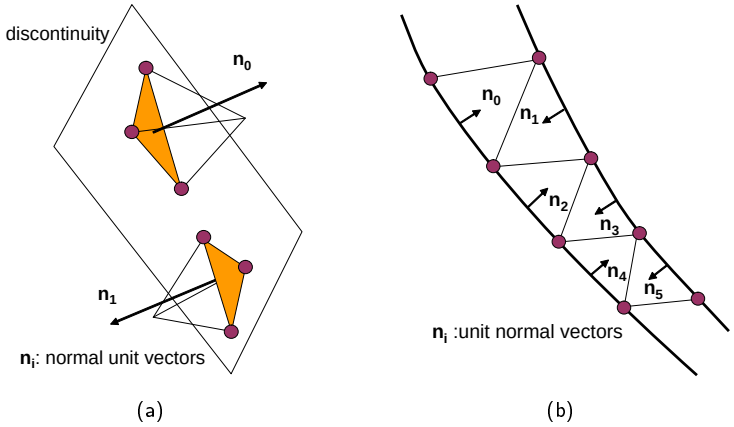
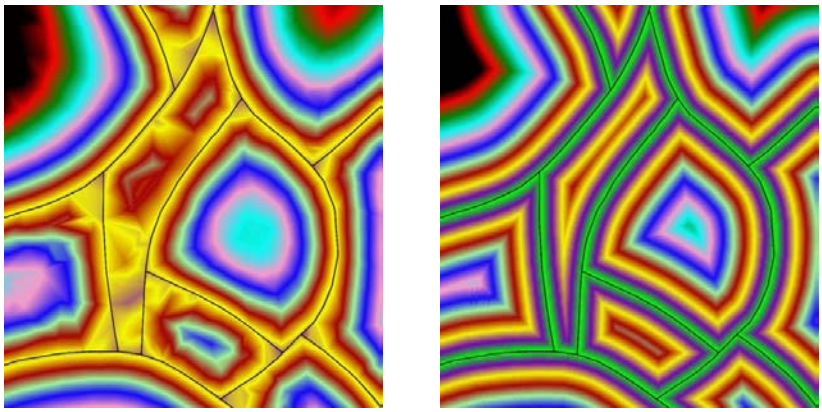


Figure 3.5: (a) shows a discontinuity and two outlined tetrahedrons on each side of the discontinuity. The vertices of the tetrahedrons faces that are coplanar with this unconformity are used to define points with a distance zero. Normal unit vectors pointing to the interior of each sub-volume are installed to define the gradient of a distance function. (b) shows two discontinuities connected only by one layer of tetrahedrons.

method to all tetrahedrons adjacent to an unconformity like a geological fault and running a DSI interpolation with the above-mentioned constraints results in an interpolated distance function approximating the distance of an arbitrary point of the unstructured domain to the nearest unconformity (fault). If the object O consists of points, each point carries the distance zero and the gradient vectors are installed in arbitrary directions on that location.

The advantage of this interpolated distance transform is the explicit definition of a distance function D on the nodes of the simplicial $3d$-complex. So this interpolated distance map can be directly used for subsequent calculations. The quality of the results obtained by this method depends to some extend on the constitution of the triangulation of the tetrahedral mesh. Let us consider two close unconformities and the gap between them is only populated by one layer of tetrahedrons. Thus, one node of a tetrahedron's edge lies on one unconformity and the second node of this edge lies on the other unconformity, see figure 3.5(b). The properties are interpolated linearly between these two nodes and thus the distance function takes zero values between the unconformities. But not only direct connections but also coarse resolution of the tetrahedral mesh can lead to undesired artifacts. Figure 3.6(a) shows a cross-section of a tetrahedral model. The fault network is highlighted with black lines. At the fault connections and in regions of close faults one can clearly see artifacts. In the interior and in regions distant from the faults the distance function is regular. An additional disadvantage of this method is the relatively high computational effort to interpolate the distance function for large grids. Many applications do not need an explicit distance map and a vision of this distance function is sufficient. The next sections explain how to visualize a distance transform on an tetrahedral mesh without the explicit computation of this function on the nodes of this mesh.



(a) (b)

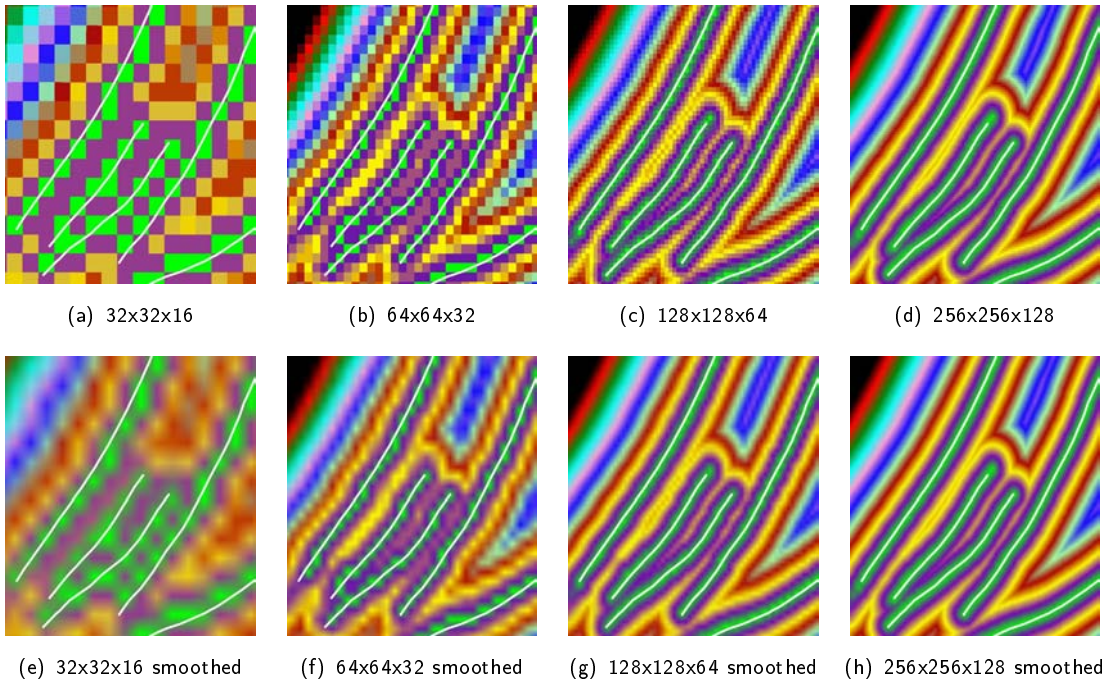
Figure 3.6: (a) and (b) show a cross-section through tetrahedral mesh. (a) illustrates a interpolated distance transform. Artifacts close to the unconformities are clearly visible. The texture-mapped distance transform of (b) does not contain any artifacts. Data by courtesy of Total.

Texture-Mapped Distance Maps

The basic idea behind this implicit method is to compute an EDT on a regular background grid and to use this background grid as a $3d$-texture map. Transferring a problem from an unstructured domain into a structured computational space is widely used for surface reconstruction [Hoppe et al., 1992] and medical image processing [Cebral et al., 2003]. For the computation of the EDT on a Cartesian grid, an adaption of [Meijster et al., 2000] for the $3d$-space introduced by [Ledez, 2002] was used. The Cartesian grid $\mathscr{G}$ is aligned to the orthogonal normal vectors $\mathbf{u}, \mathbf{v}, \mathbf{w}$ and consists of $n_u * n_v * n_w$ voxels. The magnitude of the Cartesian grids on which the EDT is computed is relatively small and takes for each dimension one of the values $n_u, n_v, n_w \in \{64, 128, 256\}$. The relation of 2^n is caused by the current implementation of OpenGL Version 1.5 [Segal and Akeley, 2003] where texture maps have to fulfill this condition. Subsequent versions of OpenGL support texture maps that may have arbitrary dimensions [Segal and Akeley, 2004]. The set of objects O has to be rasterized to a binary $3d$-image $o[n_u, n_v, n_w]$ on the domain of $\mathscr{G}$. In the following, we briefly introduce the algorithm for the computation of an EDT according to [Meijster, 2004] on $\mathscr{G}$. The algorithm to compute the EDT on a $3d$-image can be broken down in three phases. For the first two phases we decompose $\mathscr{G}$ into a series of n_w $2d$-images and compute the EDT for this reduced problem. So we can write:

$$\begin{aligned} EDT(u,v) &= MIN(i,j:\ 0 \le i < n_u \ \wedge\ 0 \le j < n_v \ \wedge\ o[i,j]:\ (u-i)^2 + (v-j)^2) \\ &= MIN(i:\ 0 \le i < n_u:\ (u-i)^2 + G(i,v)^2) \text{ with} \qquad (3.2) \\ G(i,v) &= MIN(j:\ 0 \le j < n_v \ \wedge\ o[i,j]:\ |v-j|) \qquad (3.3) \end{aligned}$$

using the notation $MIN(itr : P(itr) : f(itr))$ for the minimum of $f(itr)$ where itr ranges over all the values that fulfill the condition $P(itr)$. In the third phase, the $2d$-results are the basis for the assembly of a $3d$-EDT.



(a) 32x32x16 (b) 64x64x32 (c) 128x128x64 (d) 256x256x128

(e) 32x32x16 smoothed (f) 64x64x32 smoothed (g) 128x128x64 smoothed (h) 256x256x128 smoothed

Figure 3.7: These figures show cross-sections in a tetrahedral mesh. The visualized function illustrates the distance to the white outlined faults. The resolutions of the $3d$-texture maps is indicated. The second row was rendered using texture interpolation. Data by courtesy of Earth Decision.

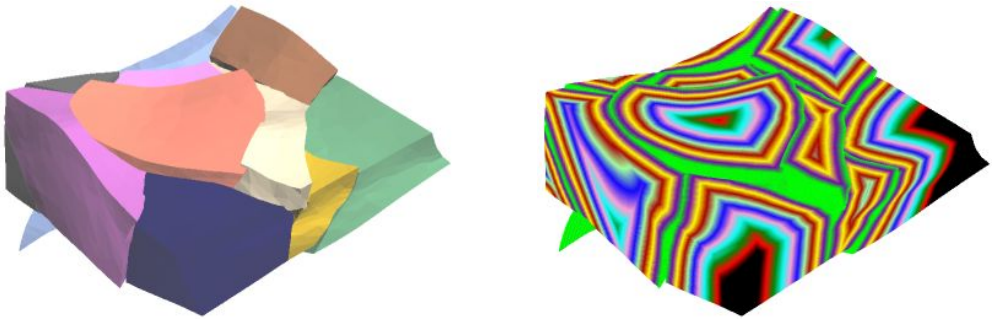
Initialization In the first phase, each column C_u of pixels $P(u,v)$ with fixed u is scanned independently. For all pixels of C_u we compute the minimum distance $G(u,v)$ of $P(u,v)$ to the pixel where C_u has a non-empty intersection with O.

$2d$-EDT In the second phase, each row R_v of pixels with fixed v is scanned separately. Constant v is used as parameter in the auxiliary function $g(i) = G(i,v)$. Together with the results of the first phase, the computation of the $2d$-EDT results in a minimization problem:

$$EDT_{2d} = MIN(i: \ \ 0 \leq i < n_u : \ \ (u-i)^2 + g(i)^2) \tag{3.4}$$

For any discrete $i = 0...n_u$ we define the function $F_i : u \mapsto (u-i)^2 + g(i)^2$ on the real interval $[0, n_u - 1]$. The graph of F_i is a parabola with its vertex at $(i, g(i))$. Proceeding this way one obtains a set of parabola. The EDT_{2d} can be evaluated as the lower envelope of this set of curves.

Extension to Three Dimensions One feature of the algorithm of [Meijster et al., 2000] is that the nd-problem can be decomposed into n $1d$-problems. The $2d$-EDTs computed in the first two phases are simply combined to a $3d$-image. This algorithm can be extended to three dimensions. The combined result is considered as a set of $2d$-images but this time aligned to the axis u and v. Applying the second phase independently to all columns C_w with fixed w of this images finally results in the desired $3d$-EDT. The outstanding properties of the texture-mapped distance maps are the high resolution without artifacts



(a) (b)

Figure 3.8: These figures show a flattened model. The flattening was performed with the GeoChron parametrization. (a) shows the fault blocks. (b) shows the same fault blocks texture-mapped with an Euclidean Distance Transform. The distance transform is based on the distance to the fault network. Data by courtesy of Total.

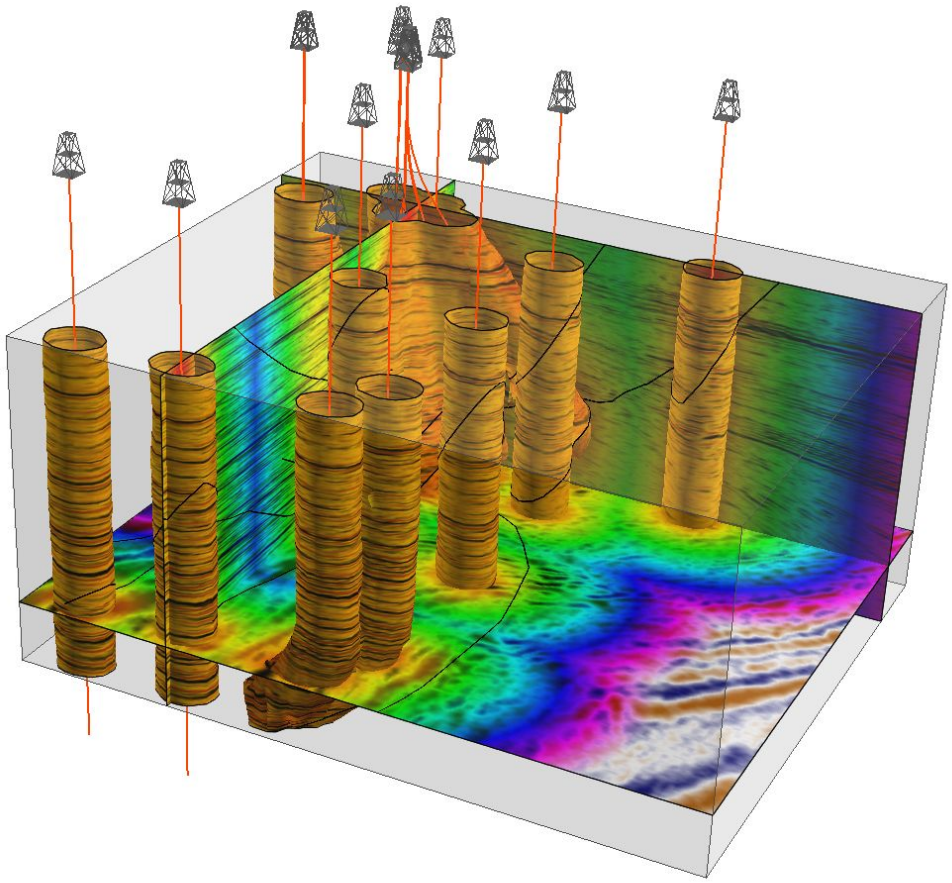


Figure 3.9: This figure shows cross-sections and iso-value surfaces extracted from a tetrahedral mesh. The iso-value surfaces are defined by an iso-distance to the wells. The distance function and seismic data are co-visualized. Fault traces are highlighted by black outlines. Data by courtesy of Earth Decision.

(see figure 3.6(b)), and their fast computation that is one order of magnitude less than the interpolated texture maps. Figure 3.7 shows cross-sections in a tetrahedral mesh. The visualized function illustrates the distance to the white outlined faults. The distance map was computed with different resolutions and the rendering was performed with and without texture interpolation. Further applications of EDTs are shown in figures 3.8 and 3.9.

3.3.3 Stratigraphic Column

A stratigraphic column is an image that assigns distinct colors to a model according to given geological time intervals. These markers can easily be combined with stratigraphic grids where the borders of a stratigraphic layer are aligned to the cell boundaries of this grid. The extension of the stratigraphic column to unstructured grids like the tetrahedral mesh – investigated in this work – is more complex. As precondition we need the definition of a geological time function on the nodes of the simplicial $3d$-complex. This is realized with the GeoChron parametrization. A naive approach is the numerical extraction of level sets of this time parametrization and relating color values to the time intervals. Due to the numerical costs and the lack of elegance this solution is not further investigated.

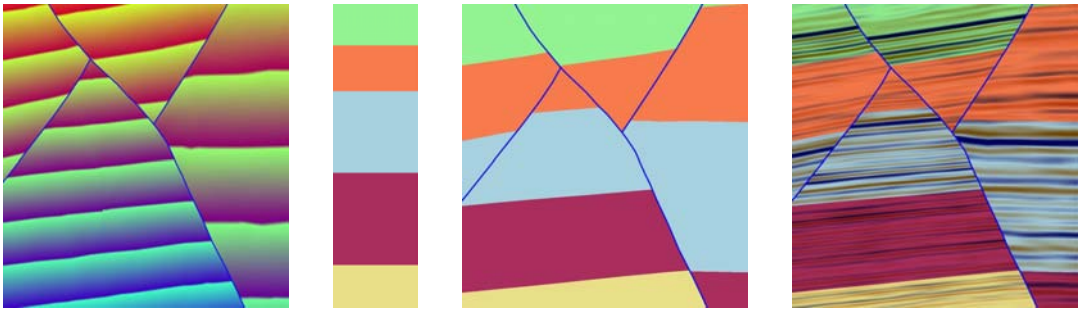


Figure 3.10: (a) shows a GeoChron time function painted with a periodic colormap. (b) shows a color-bar computed from a step function over the GeoChron time. (c) shows the model rendered with the step function. The stratigraphic layers are outlined in different colors. (d) combines the image of (c) with a $3d$-texture map extracted from a seismic cube. Data by courtesy of Total.

A stratigraphic column can be considered as a step function over geological time (see figure 3.3). Assigning the red, green and blue color channel and the alpha channel to the values of this step function, one receives a color-bar, as illustrated in figure 3.10(b). This color-bar has a constant color for each interval of the geological time and can be used as $1d$-texture map. The proper texture coordinates are the values of the time function (see figure 3.10(a)). Figure 3.10(c) shows the result of texture mapping a continuous time function with a step function. As this rendering is performed on the GPU there is no loss of performance compared to ordinary rendering. The stratigraphic column and its color values are user-defined parameters. Of course this image can be co-rendered with other information like seismic data from a $3d$-texture map (see figure 3.10(d)). Figure

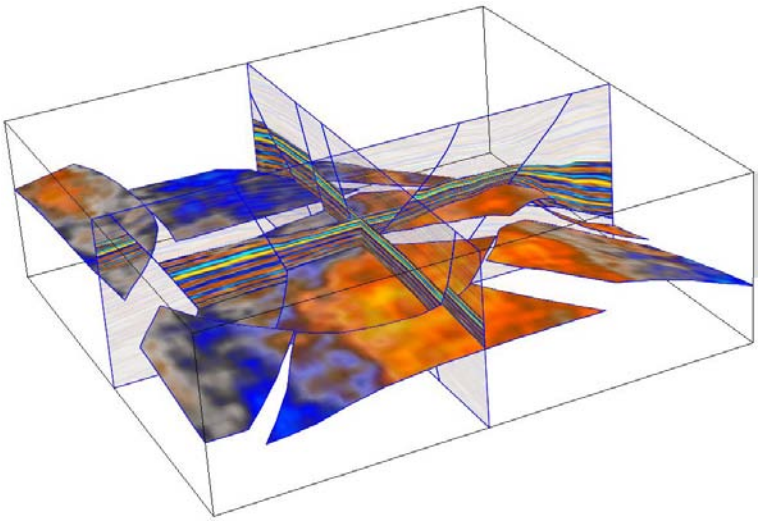


Figure 3.11: Shows two cross-sections and an iso-T surface in a tetrahedralized model. The surfaces were texture-mapped with a synthetic seismic cube and co-rendered with a stratigraphic column. One stratigraphic layer is highlighted, the other parts have a higher transparency. Data by courtesy of Total.

3.11 shows a model where a stratigraphic column was used to define the transparency and therefore distinct stratigraphic layers were highlighted.

3.3.4 Implicit Intersection, Clipping and Iso-Contouring

The previous section introduced step functions over geological time to visualize the stratigraphic column on a model. This section will generalize the concept of procedural texture maps combined with different texture coordinates. The basic ideas behind this concept will be elaborated in three steps: (1) the implicit calculation of intersections between tri-variate functions, (2) visualization of iso-contours of scalar functions on level sets and finally (3) the application of boolean CSG operations to compute clipping and penetration of implicitly defined bodies.

Implicit Intersection

The examined problem is the computation of the intersection of an iso-value surface $\mathscr{S}_A$ of one function φ_A with an iso-value surface $\mathscr{S}_B$ of a second function φ_B. An example is the outline of a geological horizon on a cross-section. The numerical solution is the application of the Marching Tetrahedron (MT) algorithm twice. In the first pass the polygons of the iso-value surface $\mathscr{S}_A$ are computed. Applying MT a second time constrained by an iso-value of function φ_B to this set of polygons results in the intersection curve. So the additional cost is the second run of MT. When the extracted iso-value surface $\mathscr{S}_A$ consists of n polygons and k of these n are intersected by $\mathscr{S}_B$, the additional effort is at least $\mathscr{O}(k+log(n))$ using an interval tree [Bajaj, 1999] or parametric octree decomposition (see section 2.3.1). This effort increases linearly for each additional intersection to

be computed. A graphical algorithm with constant complexity $\mathcal{O}(1)$ will be introduced in the following.

The basic idea is the same as applied to visualize the stratigraphic column. To visualize an intersection curve of two functions, let us draw the iso-value surface $\mathcal{S}_A$ with a step function over the domain of φ_B. This step function takes identity values for the blending function except for the value the intersection is to be computed. For the `GL_MODULATE` blending function we define a $1d$-texture map with opaque white values. Only the position of the intersection value carries the color information of the intersection curve. The normalized values of φ_B are used as texture coordinates.

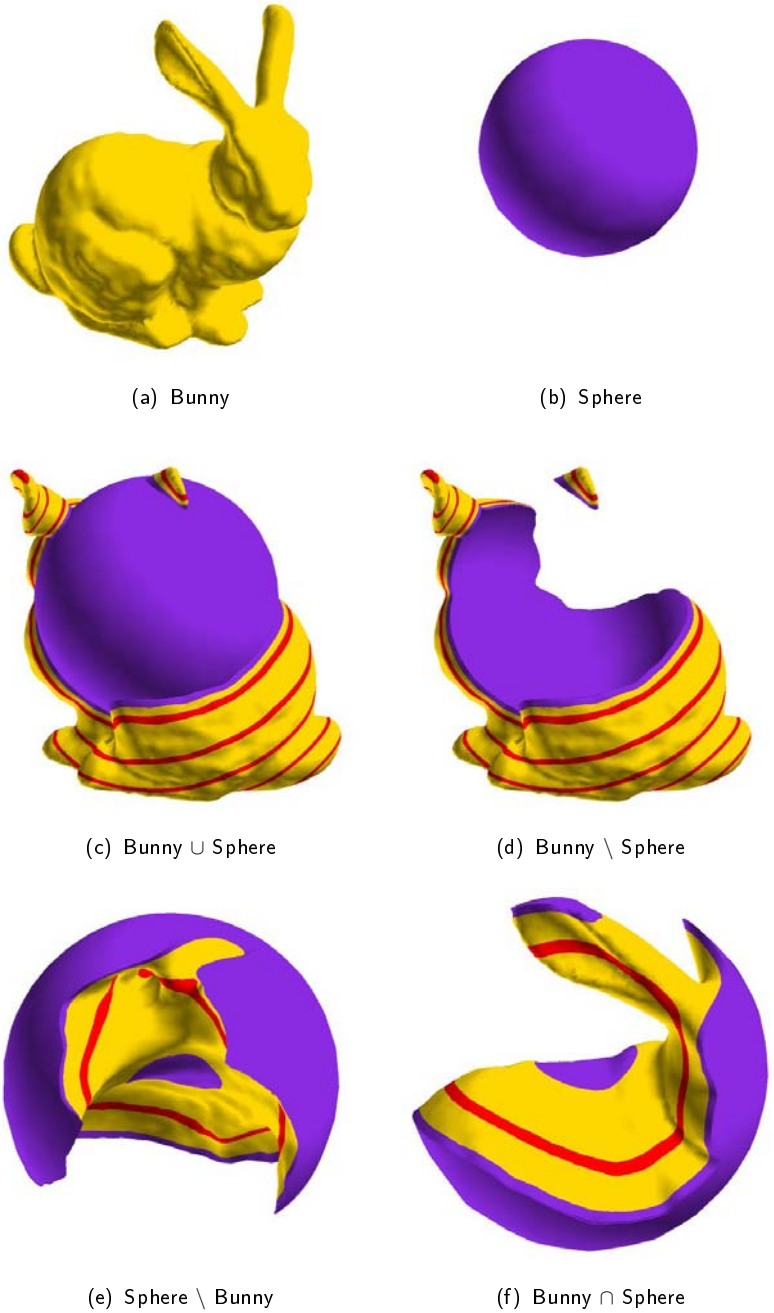
Iso-Contours

Iso-contours can be considered as a series of equidistant intersections between an iso-value surface and the implicit function the iso-contours are extracted from. Figure 3.1 shows black outlined iso-T contours painted on a cross-section that was co-rendered with a stochastic simulation. Iso-contours can be considered as a periodic step function (see figure 3.3). The computational effort only consists in the computation of the $1d$-texture map from the periodical step function and can be neglected.

Clipping of Surfaces and Penetration of Bodies – Applications to CSG

The above introduced algorithm for visualizing the intersection of two implicit surfaces is developed a little further. Especially for CSG applications, not only the intersection between two bodies but also their clipping and penetration is of great importance. Let us consider a step function over the domain of the function φ_A. This function takes zero values in the interval $[\min(\varphi_A), \beta[$ and is one in the interval $[\beta, \max(\varphi_A)]$ where β is an arbitrary value of the range of φ_A and defines the clipping threshold. The values of this step function can be treated as alpha values of a texture map, and rendering an iso-value surface of the function φ_B with this texture map will paint all parts of the iso-value surface that take values less than β of φ_A transparent. This is a graphical solution for a clipping algorithm with constant complexity $\mathcal{O}(1)$ for two implicit surfaces of arbitrary size and complexity. Figure 2.2(b) shows two cross-sections painted with a pseudo distance function. The salt dome was clipped with this algorithm. This method can easily be extended to more surfaces using more or higher-dimensional texture maps.

In CSG applications, bodies of high complexity are often defined as implicit functions. Implicit functions can be easily defined on the nodes of the tetrahedral mesh. In the following, the convention that positive values define the interior and negative values define the exterior of a body is used. The iso-value surface where the implicit function takes the value zero is the boundary surface of the body. Multitexturing techniques can be used to perform boolean operations like union, difference and intersection on two or more bodies defined by the implicit functions. Figure 3.12 shows a reconstruction of the Stanford Bunny data set and a sphere. Both bodies are defined as implicit functions with the above-mentioned constraints. The images show the results of the boolean CSG operations. For each boolean CSG operation a transfer function over the domains of the implicit function



(a) Bunny (b) Sphere

(c) Bunny ∪ Sphere (d) Bunny \ Sphere

(e) Sphere \ Bunny (f) Bunny ∩ Sphere

Figure 3.12: This series of figures shows the application of boolean operators to CSG. The Stanford Bunny (a) and a sphere (b) are defined as implicit functions on the nodes of a simplicial $3d$-complex. The CSG operations were performed using multitexturing techniques with constant complexity $\mathcal{O}(1)$. Data by courtesy of Stanford $3d$-Scanning Repository.

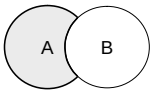
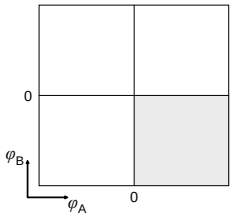
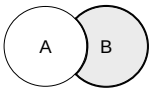
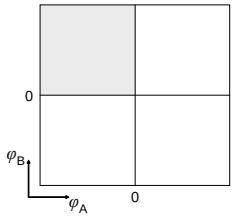
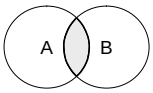
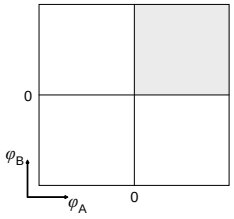
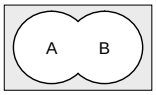
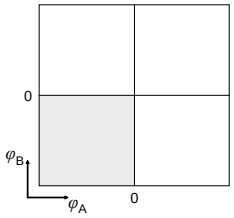
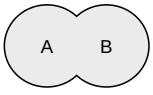
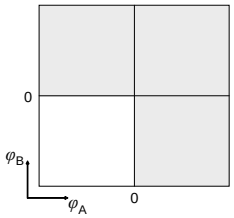
CSG operation	Look-up table for transparency	Cross-section through CSG result
$A \setminus B$ 	$\alpha = 1 \forall \varphi_A \geq 0$ AND $\varphi_B < 0$ 	
$B \setminus A$ 	$\alpha = 1 \forall \varphi_A < 0$ AND $\varphi_B \geq 0$ 	
$A \cap B$ 	$\alpha = 1 \forall \varphi_A \geq 0$ AND $\varphi_B \geq 0$ 	
$(\overline{A \cup B})$ 	$\alpha = 1 \forall \varphi_A < 0$ AND $\varphi_B < 0$ 	
$A \cup B$ 	$\alpha = 1 \forall \varphi_A \geq 0$ OR $\varphi_B \geq 0$ 	

Table 3.1: The look-up tables define the transparency (white quadrants) over the domains of the implicit functions φ_A and φ_B. φ_A and φ_B describe two bodies A and B with the convention that positive values define the interior and negative the exterior of the bodies. Each texture map is applied to the iso-value surface $\mathscr{S}_A$ and $\mathscr{S}_B$. The right column illustrates cross-sections through the result of the boolean CSG operation with additional texture mapping. φ_A defines a bunny and φ_B a sphere.

φ_A and φ_B defines the transparency of a fragment according to the value tuple of the implicit functions. The transfer functions are illustrated in table 3.1. This method can be generalized to n-implicitly defined bodies by nd-transfer functions. Again the complexity of this method only depends on the computation of the texture maps from the nd-transfer functions and is hereby constant $\mathcal{O}(1)$ for any model size and complexity.

3.4 Summary

This chapter illustrated visualization techniques with a special focus on geo-modeling applications. Geological interfaces like faults or horizons can be outlined explicitly if this information can be extracted from the topological information inherent to the tetrahedral mesh. If this information is not available, horizons can be visualized by a parametric function used for multitexturing. Multitexturing further allows the co-visualization of multiple attributes like distance maps, stratigraphic columns or any other information stored as texture map. The blending of the different attributes can be performed by the built-in algorithms of the OpenGL graphics pipeline or more elegantly by user-defined blending functions implemented in fragment shaders. A second application of multitexturing was explored to perform clipping and boolean CSG operations on the graphics hardware with constant complexity independent from model size and complexity. $3d$-GIS applications [Apel, 2004; McGaughey et al., 2004] can make use of this technique to perform high-performance conditional queries.

Chapter 4

Local Mesh Refinement

Contents

4.1 Introduction

The preceding chapters introduced visualization techniques for tetrahedral meshes with focus on geo-modeling applications. The second objective of this work is modeling of tetrahedral meshes. This chapter introduces mesh refinement of simplicial complexes as a geometric modeling tool. It reviews local densification methods for tetrahedral meshes, illustrates implementation issues and discusses their results for applications using the Discrete Smooth Interpolation (DSI) method (see section 1.1.2) like the surface reconstruction method introduced in chapter 5. Simplicial complexes used for geo-modeling are built from structural models obtained by seismic surveys and well data using the algorithms presented by Conraud [1997], Conreaux [2001] and Lepage [2003]. These methods allow variable size of the tetrahedrons by defining the density of the inserted points used by the algorithm of Bowyer [1981] and Watson [1981]. Data and requirements for tetrahedral models can change throughout the modeling workflow. Thus, the cell size of the tetrahedrons need to be adapted in order to capture local phenomena.

Many methods propose a regular subdivision of all tetrahedrons of an initial simplicial complex. These algorithms result in huge numbers of cells and also refine regions that already have sufficient resolution. Contrary to these approaches local refinement only decreases the cell size of a certain region of interest. The main difficulty of local refinement methods is to preserve the conformity of the mesh. A conformal $3d$-simplicial complex is a mesh where the intersection of any two tetrahedrons is either a face, an edge, a point or is empty.

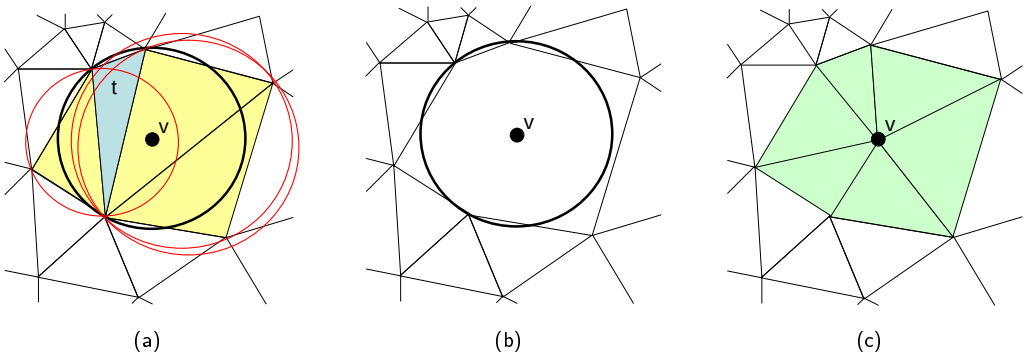


Figure 4.1: (a) shows a triangle t that is split by Delaunay refinement. Therefore a new vertex v is inserted at the center of the circumscribing circle of the triangle. The yellow shaded triangles are violating the Delaunay criterion as the inserted point lies inside their circumcircle and are deleted (b). The generated cavity is re-triangulated as shown in (c).

4.2 Tetrahedral Mesh Refinement Methods

4.2.1 Classification

One can distinguish three main classes of tetrahedral mesh refinement methods:

Point Insertion New points are inserted into the tetrahedral mesh and hereby the region around the point is re-triangulated honoring the Delaunay criterion. This method will be called Delaunay refinement throughout the rest of this chapter.

Edge Bisection An edge of the tetrahedral mesh is intersected and all tetrahedrons sharing this edge have to be split into two simplices.

Templates A group of templates defines schemas how a certain tetrahedron and its neighbors have to be decomposed.

Algorithms using GPU acceleration like [Pascucci, 2004; Boubekeur and Schlick, 2005] are not considered since they only produce visual results and are not applicable for numerical application performed on the CPU.

Delaunay Refinement Using Point Insertion

Point insertion algorithms are based on Delaunay tessellation. The Delaunay criterion for tetrahedral meshes states that there is no point of any other tetrahedron in the circumsphere of a tetrahedron. Mesh refinement based on the Delaunay criterion was proposed in two dimensions by Chew [1987] and Ruppert [1995] and extended into three dimensions by Shewchuk [1998]. Delaunay is only a criterion to define connectivity and is not a tessellation algorithm. To perform Delaunay triangulation we use the algorithm of Bowyer [1981] and Watson [1981]. This algorithm uses the following steps to perform Delaunay triangulation:

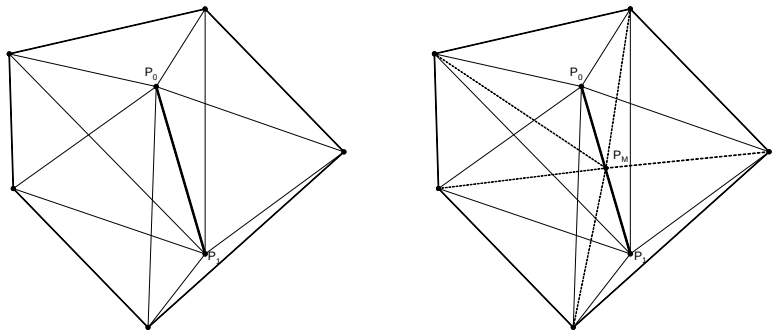


Figure 4.2: (a) shows the intial state with an edge P_0P_1 and five tetrahedrons sharing this edge. When this edge is bi-sected, the tetrahedrons have to be split into two tetrahedrons to maintain the conformity of the mesh (b).

1. A new point is inserted at the center of the circumsphere or at the barycenter of a tetrahedron.
2. The tetrahedrons whose Delaunay criterion is violated by the new point are removed from the mesh.
3. The facets of the produced cavity are connected with the inserted point to build the new tetrahedrons.

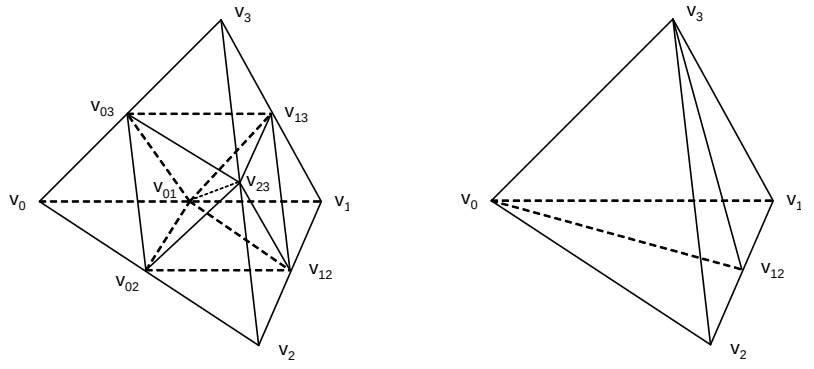
An obvious drawback of the point insertion method is that the element size only decreases slightly while for each point that is inserted a certain neighborhood is affected. Figure 4.1 shows Delaunay refinement for a $2d$-triangular mesh. The extension to a $3d$-tetrahedral mesh is straight-forward.

Edge Bi-Section Refinement

Edge Bi-section in two dimensions divides a triangle into two triangles by intersecting an edge of the triangle. Plaza and Carey [1996] present an extension of this algorithm to three dimensions and its application on tetrahedrons. Whenever an edge of the tetrahedral mesh is intersected, all tetrahedrons of the mesh sharing this edge have to be split into two tetrahedrons (see figure 4.2). Arnold et al. [2000] propose a recursive refinement scheme to achieve conformity for local mesh refinement. The implementation realized for this work uses longest edge intersection together with templates to realize a conformal mesh. A basic version of edge Bi-section is easy to implement but its disadvantage is the lack of a guarantee for quality meshes.

Template-Based Octa-Section Refinement

Plaza and Rivara [2003] proposed a consecutive edge Bi-section together with predefined partition templates to decompose a tetrahedron into eight simplices. Templates define



(a) Sub 8 refinement, six edges were split

(b) Sub 2 refinement, one edge was split

(c) Sub 4 refinement type 1, two edges that do not share the same face were split

(d) Sub 4 refinement type 2, three edges that share the same face were split

Figure 4.3: (a) shows the principal template to decompose a tetrahedron into eight tetrahedrons. Figures (b) to (d) illustrate additional templates used to guarantee conformity. Templates after [Liu and Joe, 1996].

a certain decomposition pattern to split triangles or tetrahedrons, respectively. Liu and Joe [1996] and L. Endres [2003] proposed to decompose a tetrahedron into eight similar simplices (see Figure 4.3(a)). Therefore each face is split into four similar triangles. This refinement pattern produces four tetrahedrons and one octahedron in a first step. To decompose the octahedron into four tetrahedrons one has to choose a split edge. This decision should be based on quality measures. If the refinement has local character, additional templates are needed to obtain a conforming mesh (see figure 4.3(b) to 4.3(d)).

The implementation realized for this work uses the decomposition templates shown in figure 4.3. All simplices designated for decomposition are treated with the decomposition template from figure 4.3(a). The boundary of the decomposed regions must also be refined to achieve a conforming mesh. The template that has to be applied to a simplex is defined by the number of edges that were split. The templates from figure 4.3 only define patterns for one edge split (figure 4.3(b)), two edge splits on edges that do not share the same face (figure 4.3(c)), three edge splits on edges that share the same face (figure 4.3(d)) and the principal decomposition where all six edges were split (figure 4.3(a)). Any other configuration has to be deduced from a given decomposition template. A simplex with

two edge splits where the edges share the same face is decomposed into four simplices according to the template from figure 4.3(d). Simplices with four or five edge splits are decomposed into eight tetrahedrons according to the template from figure 4.3(a). This procedure is needed to conform the mesh and can be achieved by applying a recursive algorithm shown in listing 4.1. The recursion always terminates with a worst case where all tetrahedrons of a simplicial complex were regularly subdivided into eight simplices. This case never occurred in our models.

Listing 4.1: Pseudo code for conform_mesh

```
begin conform_mesh

  bool splitted2sub8 ← true
  bool splitted2sub4 ← true
  bool splitted ← false

  do
    splitted2sub8 ← false
    for all tetrahedrons T_i ∈ Ω_conf
      if T_i has 4 or 5 split edges
        sub_8( T_i )   // decompose T_i into eight tetrahedrons
        splitted2sub8 ← true
        splitted ← true
      end if
    end for
  while( splitted2sub8 == true )

  do
    splitted2sub4 ← false
    for all tetrahedrons T_i ∈ Ω_conf
      if T_i has 2 split edges
        if split edges share the same face
          sub_4_type_2( T_i )  // decompose T_i into four tetrahedrons of type 2
          splitted2sub4 ← true
          splitted ← true
        end if
      end if
    end for
  while( splitted2sub4 == true )

  if splitted == true
    conform_mesh
  end if
end conform_mesh
```

4.2.2 Results

The three refinement methods described in section 4.2.1 were implemented for the G◯CAD tetrahedral mesh. The surface reconstruction algorithm introduced in chapter 5 makes extensive use of DSI on $3d$-simplicial complexes. This algorithm interpolates an implicit function $\varphi(x,y,z)$ on a tetrahedral mesh. The iso-value surface $\mathscr{S}$ where $\varphi(x,y,z) = 0$ represents the reconstructed surface. Of course, the precision of the reconstruction depends on the resolution of the tetrahedral mesh. This reconstruction method was used to test the introduced refinement methods.

Test Method

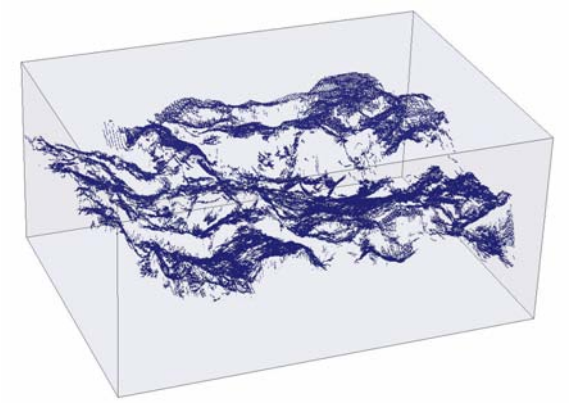
The point set to reconstruct was a salt-top surface picked (377,384 points) from seismic data (see figure 4.4(a)). Starting with a tetrahedral mesh (see figure 4.4(b)) with a coarse resolution (4,841 tetrahedrons) covering the domain of the point set, this simplicial complex was refined step-by-step until a given error bound was reached. For each point of the input data an unsigned error estimate as described in section 5.3.3 was computed. On the initial grid from figure 4.4(b) a first approximation of the reconstructed surface was computed (see figure 4.4(c)). Assuming an error bound of 500, 167,650 of the input data points were exceeding this error bound. The tetrahedral mesh was refined at the location of the erroneous input points. After each refinement the reconstruction was performed again, the error estimate was re-computed and the erroneous points were used for defining the regions for the next refinement level.

Reconstruction Convergence

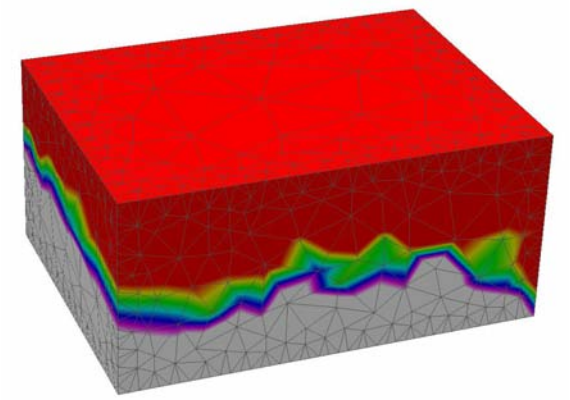
The refinement was stopped when less than 1% of the input data points were violating the assumed error bound. Using Delaunay refinement, nine refinement levels were needed and resulted in a tetrahedral mesh with 37,188 simplices. Bi-section converged much faster than Delaunay refinement. Already after four refinement levels only 3,307 points violated the error bound on a tetrahedral mesh with 48,310 simplices. The reconstruction converged even faster with Octa-section refinement. Only three refinement levels were needed but the Octa-section algorithm produced a tetrahedral mesh with 97,091 simplices. Figure 4.5 shows cutaways of the tetrahedral meshes that were refined under the above-mentioned conditions. The convergence results are illustrated in figure 4.6.

Mesh Quality

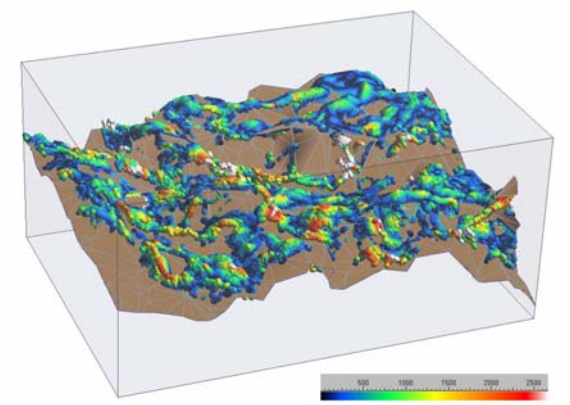
Numerical methods strongly depend on the quality of the tetrahedral mesh. Berzins [1999] discussed mesh quality issues for finite element methods and Shewchuk [2002] introduced several quality measures for discrete linear elements. To quantify the quality of the densified meshes produced by the introduced algorithms two measures were applied:



(a) 377,384 points picked from seismic data, representing a salt-top surface.

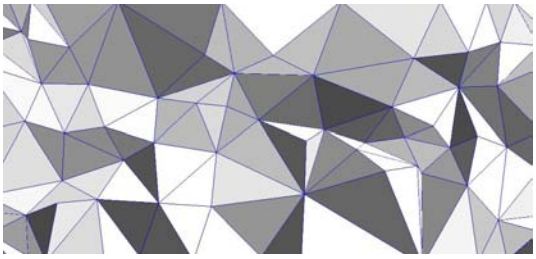


(b) Initial $3d$-simplicial complex consisting of 4,841 tetrahedrons.

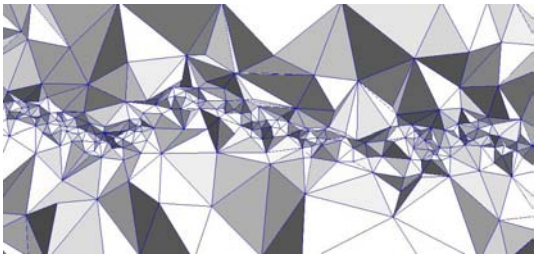


(c) Surface reconstructed from points (a) using the intial solid (b). Input data points are painted with an error function describing the interpolation error. Error color bar is included into the graphic.

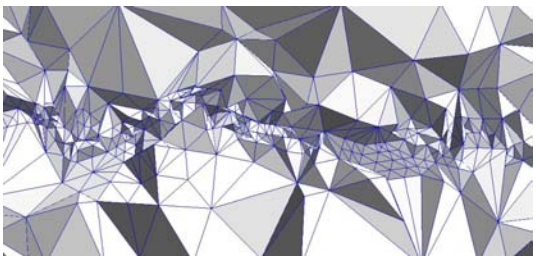
Figure 4.4: This series of figures shows the point set (a) obtained by seismic data to be reconstructed on a tetrahedral grid (b). (c) shows a first approximation of the reconstructed surface based on the tetrahedral mesh from (b). Data by courtesy of Shell.



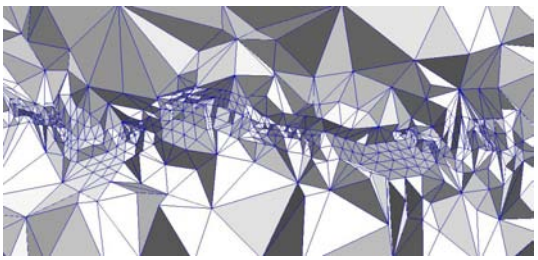
(a) Initial tessellation (4,841 tetrahedrons), ISLE (min:0.20, max:0.97, mean:0.62), CSSE (min:0.64, max:1.83, mean:0.96)



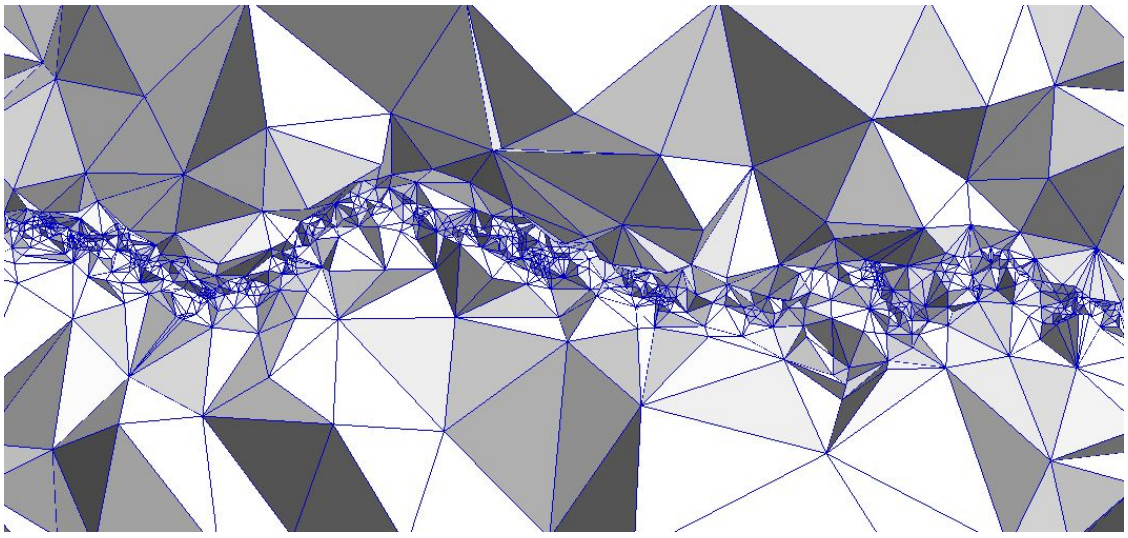
(b) Tessellation after nine levels Delaunay refinement (37,188 tetrahedrons), ISLE (min:0.01, max:0.98, mean:0.56), CSSE (min:0.62, max:9.8, mean:1.09)



(c) Tessellation after four levels Bi-section refinement (48,310 tetrahedrons), ISLE (min:0.02, max:0.97, mean:0.49), CSSE (min:0.63, max:59.97, mean:1.56)



(d) Tessellation after three levels Octa-section refinement (97,091 tetrahedrons), ISLE (min:0.04, max:0.99, mean:0.48), CSSE (min:0.62, max:30.16, mean:1.55)



(e) Tessellation after 13 levels of hybrid refinement using Delaunay (11 levels) and Octa-section (two levels) method. (265,073 tetrahedrons), ISLE (min: 0.01, max: 0.99, mean: 0.50), CSSE (min: 0.62, max: 75.01, mean: 1.47)

Figure 4.5: These figures show cutaways of a tetrahedral mesh. (a) shows the initial mesh before refinement. Densification of the meshes (b) to (d) was stopped when an assumed numerical resolution was reached. (e) shows a hybrid refinement using Delaunay and Octa-section method. Hereby a strong gradient in cell size can be achieved. Data by courtesy of Shell.

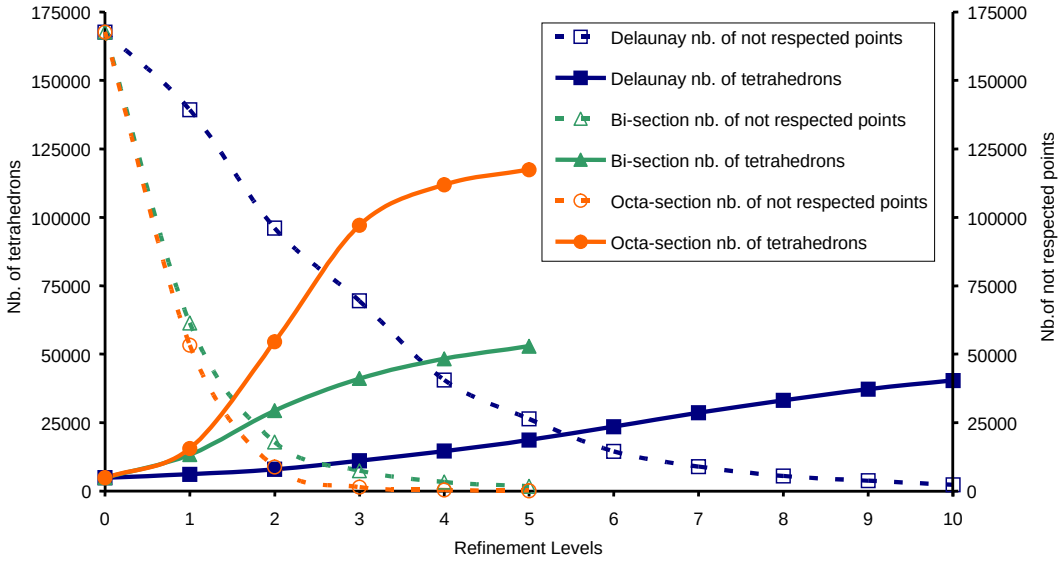
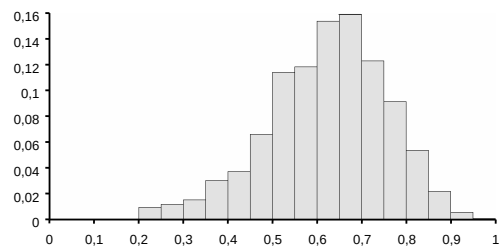


Figure 4.6: This diagram illustrates the number of points violating the error bound and the number of simplices of the tetrahedral mesh depending on the number of refinement levels. Less than 1% of the input data points were violated after nine refinement levels with Delaunay, four refinement levels using Bi-section and three refinement levels applying the Octa-section algorithm. Honoring the assumed error bound Delaunay produced 37,188, Bi-section 48,310 and Octa-section 97,091 tetrahedrons.

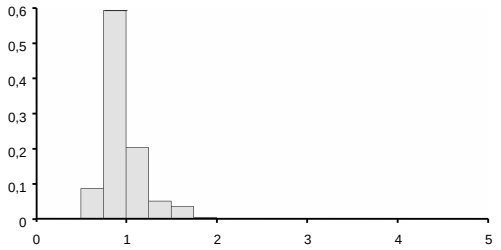
Inscribed Sphere to Longest Edge (ISLE) ratio The ISLE ratio is the built-in quality measure in GOCAD. This measure has to be maximized to increase mesh quality.

Circumscribed Sphere to Shortest Edge (CSSE) ratio The CSSE ratio is derived from the quality measure circumradius to shortest edge for triangular meshes and is used by several authors like Miller et al. [1995], Shewchuk [1998] or Alliez et al. [2005]. It can be considered as the most universal quality measure. The disadvantage of the CSSE measure is that it cannot detect slivers. Slivers are tetrahedrons whose four vertices are nearly coplanar lying on an equator of a sphere, but this type of simplices are taken into account by the ISLE ratio.

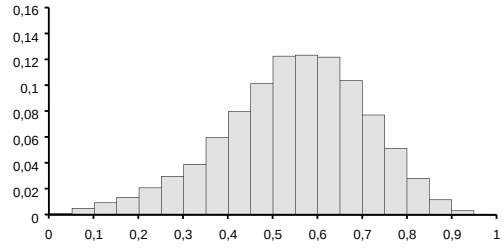
To compare the quality of the refined tetrahedral meshes, the meshes with an interpolated function that violated less than 1% of the input data points were taken. The Delaunay refined mesh had 37,188 tetrahedrons, the Bi-section refined mesh had 48,310 tetrahedrons and the Octa-section refined mesh had twice the number of tetrahedrons: 97,091. Applying both ratios the Delaunay refined mesh showed the best distributions of the quality measures. Bi-section and Octa-section produced the same quality in regard to the mean values of ISLE and CSSE ratio, but Octa-section produced less degenerated cells compared to Bi-section. At the same time the Octa-section refined mesh had twice the amount of simplices the Bi-section refined complex had and had a much higher gradient in cell size. The distribution of the quality measures are illustrated in figure 4.7. An experimental proof showed that the matrix version of DSI produces reliable results with an ISLE ratio greater than 0.01 and a CSSE ratio less than 80. Finite element methods are much more sensitive to cell quality as they depend on the condition number of the Jacobian matrix and its related matrices that are influenced by the geometry of a simplex.



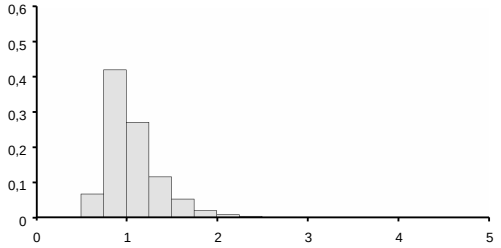
(a) ISLE ratio distribution of the initial tetrahedral mesh with 4,841 tetrahedrons (min:0.20, max:0.97, mean:0.62).



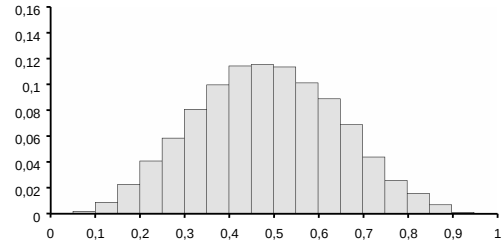
(b) CSSE ratio distribution of the initial tetrahedral mesh with 4,841 tetrahedrons (min:0.64, max:1.83, mean:0.96).



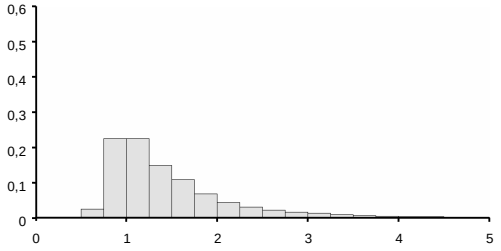
(c) ISLE ratio distribution after nine levels of Delaunay refinement with a total of 37,188 tetrahedrons (min:0.01, max:0.98, mean:0.56).



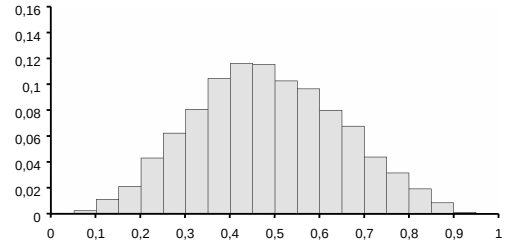
(d) CSSE ratio distribution after nine levels of Delaunay refinement with a total of 37,188 tetrahedrons (min:0.62, max:9.8, mean:1.09).



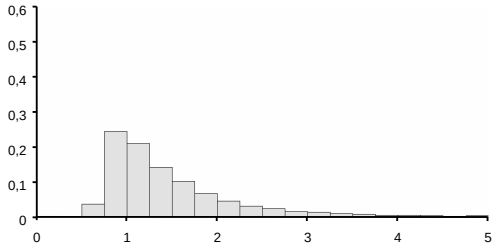
(e) ISLE ratio distribution after four levels of Bisection refinement with a total of 48,310 tetrahedrons (min:0.02, max:0.97, mean:0.49).



(f) CSSE ratio distribution after four levels of Bisection refinement with a total of 48,310 tetrahedrons (min:0.63, max:59.97, mean:1.56).



(g) ISLE ratio distribution after three levels of Octasection refinement with a total of 97,091 tetrahedrons (min:0.04, max:0.99, mean:0.48).



(h) CSSE ratio distribution after three levels of Octasection refinement with a total of 97,091 tetrahedrons (min:0.62, max:30.16, mean:1.55).

Figure 4.7: The left column shows the distribution of the ISLE ratio and the right column shows the CSSE ratio of the different refinement methods after a certain number of refinement levels when less than 1% of the data points were violating a given error bound. All histograms were normalized.

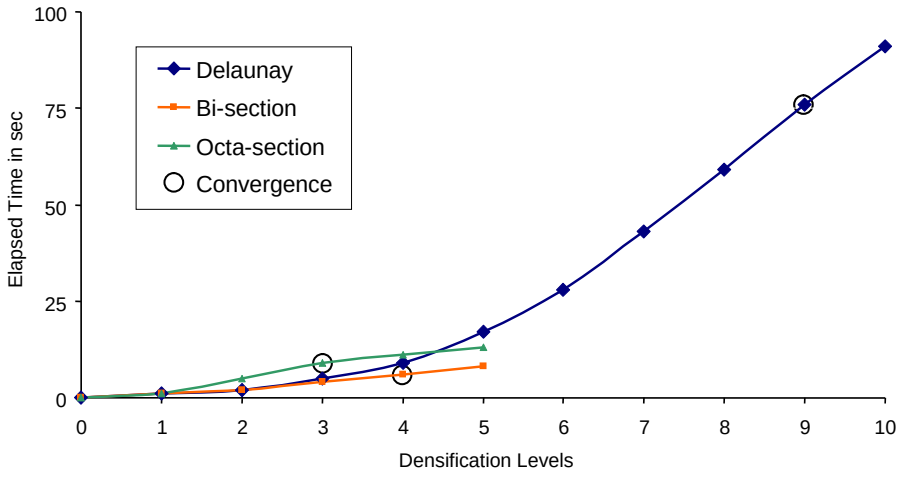


Figure 4.8: This diagram illustrates the accumulated computation time of tetrahedral mesh refinement methods depending on the refinement level.

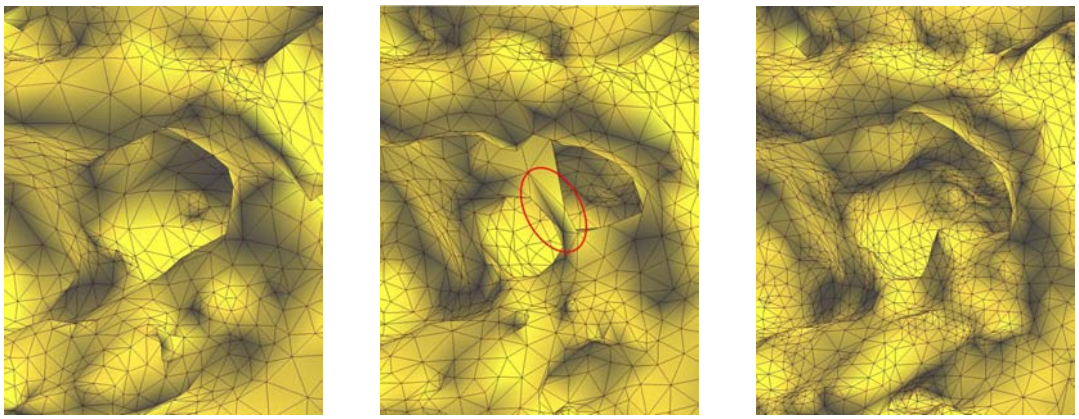
Benchmarks

This section compares the computational costs of the three refinement methods. Two criteria can be compared. The first one is the number of refinement levels needed until reconstruction convergence is reached. Octa-section reached convergence after three, Bi-section after four and Delaunay after nine levels. The second criterion is the accumulated computation time for the refinement methods until reconstruction convergence is reached. Bi-section needed six seconds, Octa-section nine seconds and Delaunay refinement 76 seconds[4] (see figure 4.8). These values only represent the computation of the refinement of the tetrahedral mesh. The overall performance is application dependant. The reconstruction process is several orders of magnitude more complex and dominates the entire process. Thus, Octa-section refinement method would be – due to the smallest number of refinement levels – the most efficient method.

DSI Performance

The previous sections handled numerical quality measures. Another important issue is the quality of the results of DSI. Figure 4.9 shows cutaways from a complex region of a surface interpolated on a tetrahedral grid using DSI. Iterative mesh refinement using the introduced methods was applied to enhance the numerical resolution of the tetrahedral mesh. One can clearly see the artefact (bridge) introduced by the Bi-section method (see figure 4.9(b)). Delaunay and Octa-section refinement produced the same topology but due to the higher amount of simplices the surface generated by the Octa-section refinement is more detailed than the surface based on Delaunay refinement (see figures 4.9(a) and 4.9(c)).

[4] All benchmarks were performed on a mobile Pentium 4 CPU with 3.06 GHz, 1 GByte of RAM and a NVidia GeForce FX Go5200 with 64 MByte memory



(a) Delaunay refinement (9 levels) (b) Bi-section refinement (4 levels) (c) Octa-section refinement (3 levels)

Figure 4.9: This series of pictures shows top views from a cutaway of a surface reconstructed on a tetrahedral mesh using the DSI method. The reconstructions were performed when less than 1% of the input data points were violating a given error criterion.

Discussion

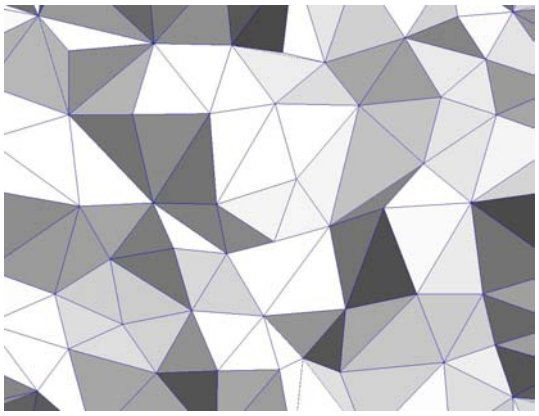
The Delaunay refinement produces the mesh with the smallest number of tetrahedrons and the best quality. But Delaunay is also by far the slowest method. The reconstruction converged after nine refinement levels and the accumulated computation time for the mesh refinement without the reconstruction costs is one order of magnitude bigger than the results of the Bi-section or the Octa-section method.

The Bi-section algorithm produces the reconstructed surface after four refinement iterations on a mesh with 48,310 tetrahedrons. The overall process is one order of magnitude faster than Delaunay but the reconstructed surface shows several undesired artifacts. Further Bi-section has an high maximum for the CSSE ratio that has to be minimized to maximize cell quality.

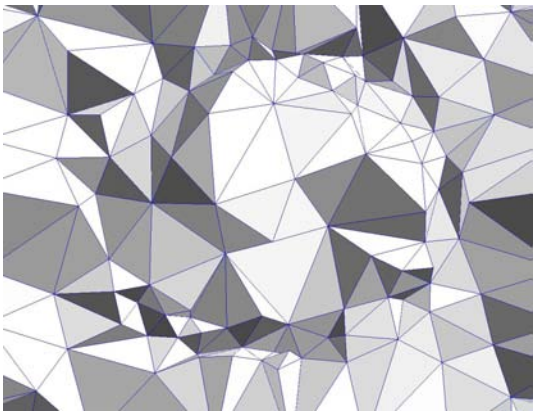
Octa-section refinement after three refinement levels already results in a high quality reconstruction but produces at the same time a mesh with twice the number of simplices compared to Delaunay or Bi-section. The quality of the tetrahedral mesh is slightly better than the mesh produced by Bi-section but has much more simplices and a bigger cell size gradient.

If a rapid result is desired (e.g. prototyping) and the numerical method can handle a huge number of simplices Octa-section is an interesting method. Further, Octa-section refinement can grant identic mesh topology beyond unconformities.

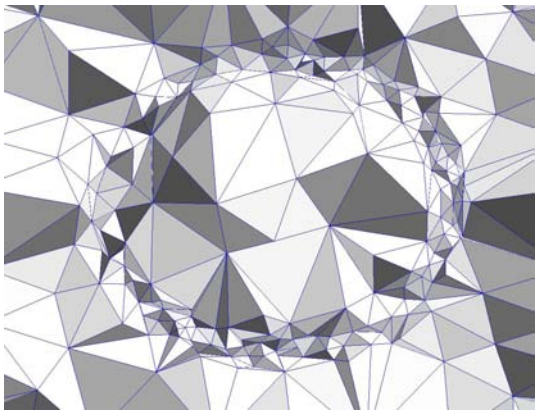
Delaunay refinement is a robust method when computation time and the number of iteration levels is less important than minimizing the size of the tetrahedral mesh and maximizing its quality. The size of the tetrahedral mesh is a crucial parameter for the majority of applications. Hereby, Delaunay refinement is the preferred method. Figure 4.10 shows the iterative local refinement using the Delaunay refinement method.



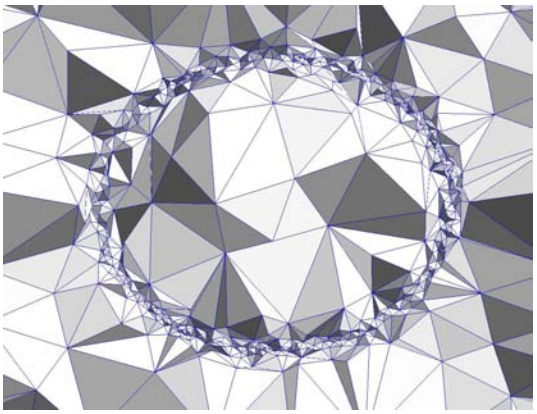
(a) Initial tetrahedral mesh (4,413 simplices), ISLE (min: 0.21, max: 0.98, mean: 0.65), CSSE (min: 0.64, max: 1.66, mean: 0.90)



(b) Three levels of Delaunay refinement (7,163 simplices), ISLE (min: 0.04, max: 0.94, mean: 0.56), CSSE (min: 0.63, max: 2.28, mean: 1.04)



(c) Six levels of Delaunay refinement (16,719 simplices), ISLE (min: 0.05, max: 0.95, mean: 0.54), CSSE (min: 0.63, max: 3.33, mean: 1.11)



(d) Ten levels of Delaunay refinement (70,732 simplices), ISLE (min: 0.02, max: 0.95, mean: 0.51), CSSE (min: 0.64, max: 4.40, mean: 1.20)

Figure 4.10: This series of figures shows cutaways from a tetrahedral mesh that was iteratively refined using the Delaunay refinement method. The refinement was performed locally on a contour of an ellipsoid.

4.3 Summary

This chapter compared three tetrahedral mesh refinement methods to improve the numerical resolution of simplicial complexes for DSI applications. Point insertion honoring the Delaunay refinement method produces the smallest meshes (in regard to the number of simplices) for an assumed numerical resolution. The mesh size is a crucial parameter and hereby the much higher computational effort is justified. For fast prototyping Octa-section is an interesting method. Bi-section converges slower than Octa-section and produces more DSI artifacts at the same time. Of course one can also combine different methods. Figure 4.5(e) shows a cutaway from a tetrahedral mesh that was locally densified using 11 levels of Delaunay and two levels of proceeding Octa-section refinement. Applying Octa-section in already Delaunay refined regions reduces the mesh bloat and increases extremly the numerical resolution in bounded regions.

A further field of exploration would be a postprocessing after the decomposition process. Freitag and Ollivier-Gooch [1996] and Cutler et al. [2004] compared and introduced several methods for tetrahedral mesh improvement. One possible mesh improvement technique is the introduced Delaunay refinement method. Therefore, a new point is inserted at the barycenter or the center of the circumsphere of a degenerated tetrahedron. Thus, the degenerated tetrahedron and some neighboring cells are violating the Delaunay criterion and are replaced by cells with better shape factors. As the matrix DSI method is very robust even on degenerated meshes this was not further pursued. Finally, one has to mention that there may be cases where a higher resolution of the tetrahedral mesh does not lead to a better interpolation result. This is always the case when noise in the input data can lead to ambiguous results.

The introduced mesh refinement methods for tetrahedral meshes were evaluated for the surface reconstruction algorithm that is introduced in the following chapter. This surface reconstruction method makes intensive use of local mesh refinement to enhance the numerical resolution of a given tetrahedral mesh.

Chapter 5

Surface Reconstruction

Contents

5.1 Introduction

The previous section introduced geometrical modeling of tetrahedral meshes to enhance numerical resolution and for mesh optimization. These techniques are widely used in this chapter. In this chapter implicit modeling of simplicial complexes is introduced. In a first approach an implicit function is computed on the nodes of the tetrahedral mesh so that a distinct iso-value surface of this function represents complex geological interfaces like horizons or salt-top surfaces. Hereby a semi-automatic surface reconstruction method is defined. Reconstructing surfaces from scattered point clouds is an omnipresent problem in computational geometry, computer graphics, computer aided design (CAD) and geo-modeling [Ledez, 2003]. These point clouds are often the output of $3d$-range scanners, physical measurements or simulations. The main objectives of this work are the reconstruction of geological objects like horizons (see figure 5.2(b)) or salt interfaces (see figures 5.1 and 5.16). The data consists of n points $p_i \in \mathbb{R}^3$ with $i \in \{1,...,n\}$. These points are just unorganized triples of x,y,z values and no other information like topology is available. The challenge is reconstructing a surface S that interpolates the set of sample points. There is no unique solution to this problem. In addition, the data sets to be reconstructed are in general not well-conditioned. They may be scattered or not evenly spaced; they can be very large, and they may have holes where no data is available, discontinuities and boundaries. The method – developed in this work – performs correctly under

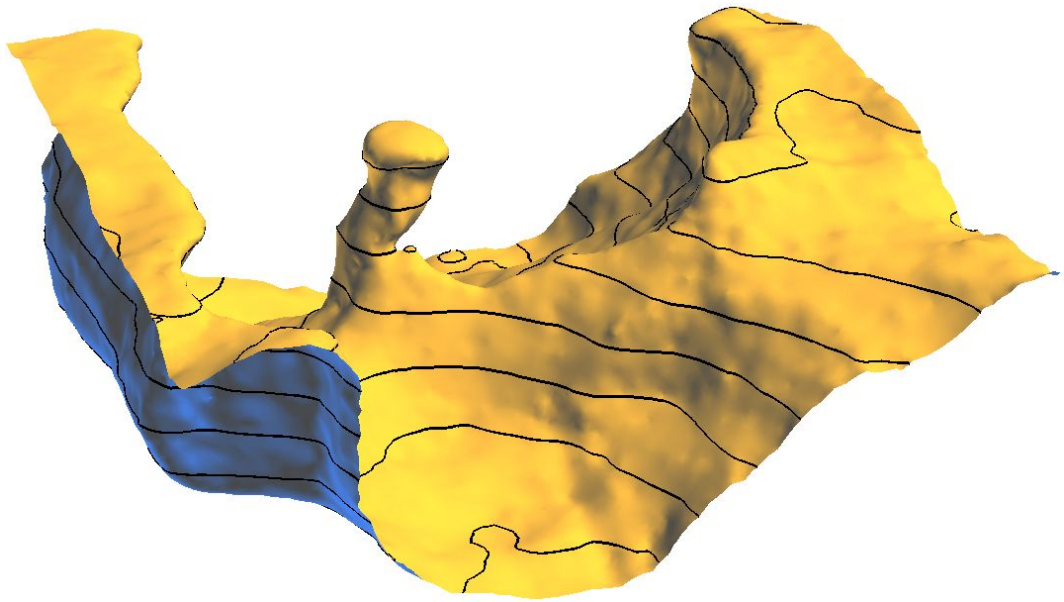
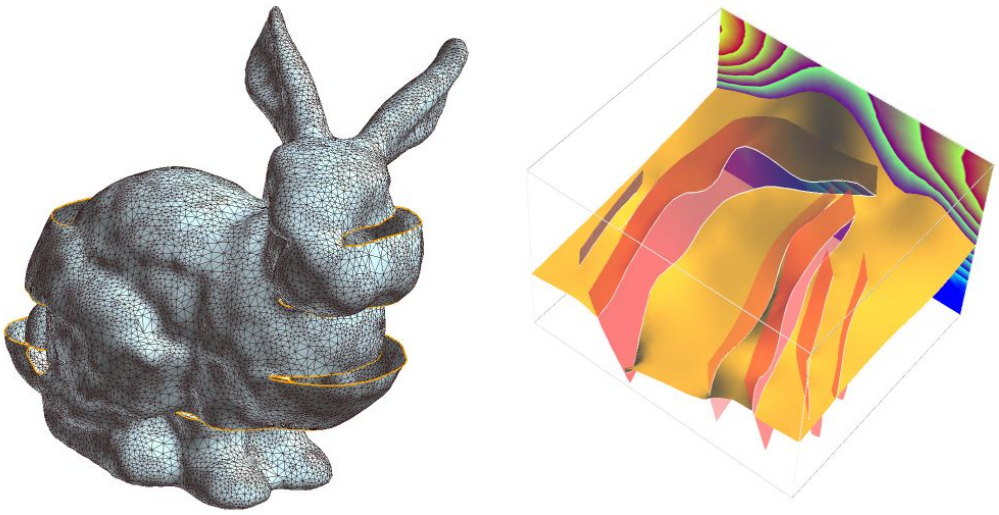


Figure 5.1: This figure shows a salt dome (47,992 triangles after beautification) reconstructed from 8,893 points. Input data is from a seismic survey where several groups of curves were picked. The reconstruction was performed on a tetrahedra tessellated volume (179,863 cells) in 247 seconds. Data by courtesy of Shell.

all aforementioned difficult conditions. It is especially able to automatically fill holes and respect discontinuities at the same time.

The introduced approach creates a piecewise linear reconstruction of the point set. In order to achieve this goal, the point set is first embedded into a $3d$-simplicial complex. The tetrahedral mesh has an adaptive resolution with huge tetrahedrons in regions distant from the point set and very small tetrahedrons close to the point data. Compared to voxelized volumes, this adaptive method allows much higher resolution in zones of interest. Further, regions with too low numerical resolution resulting in high interpolations errors can be refined iteratively. Beyond high resolution, the applied volume tessellation method also supports the definition of discontinuities. Discontinuities allow two outstanding features. First, they enable the reconstruction of very close and parallel surface patches like the Stanford bunny's ears [Turk and Levoy, 1994] (see figure 5.4(a)). Introducing an unconformity in the tessellated model between the two regions that form the parallel surface patches prevents false interconnection of these separated parts and allows the topological genus of the surface to be preserved (no handles). Second, discontinuous and bounded surfaces – which often occur in natural sciences and engineering applications – can be modeled (see figure 5.2).

In the following, an implicit function $\varphi(x,y,z)$ is interpolated on the tetrahedral mesh. The data points are boundary constraints with the value $\varphi(x,y,z) = 0$. In addition, a constraint $\varphi(x,y,z) > 0$ on one side of the point set and a constraint $\varphi(x,y,z) < 0$ on the other side is defined. Compared to the constraints of [Turk and O'Brien, 2002], that set a constant value with inverted sign for the interior and exterior of a closed surface, the developed constraints do not have an influence on the magnitude of the gradient of the



(a) (b)

Figure 5.2: (a) shows a Stanford bunny with discontinuities reconstructed from 8,171 points. The discontinuities are marked yellow. Data by courtesy of Stanford $3d$-Scanning Repository. (b) illustrates a reconstruction of a geological horizon (yellow). Discontinuities are introduced by several faults (red). The cross-section shows the values of an implicit function on a tetrahedral mesh, from which the horizon was reconstructed. Data by courtesy of Earth Decision.

implicit function. Consequently, their positions have a much smaller effect on the interpolation result and only very few of these inequality constraints are needed. The interpolation is performed using the Discrete Smooth Interpolation (DSI) method (see section 1.1.2). The complexity of this method does not depend on the number of input data points but on the number of nodes of the tetrahedral mesh, whereas the methods based on Radial Basis Functions [Savchenko et al., 1995; Turk and O'Brien, 1999; Morse et al., 2001] suffer from high complexity associated to the number of points to re-sample. In the whole workflow, no normals of the point set have to be computed, which is problematic and error-prone especially for regions with high curvature or adjacent surface parts (e.g. very thin salt bodies (see figure 5.15)). In a nutshell the contributions of this reconstruction method are the following:

- Reconstruction of ill-conditioned point sets without computing normal vectors
- Volumetric approach based on an unstructured $3d$-simplicial complex with adaptive resolution and iterative refinement
- Discontinuities in the tetrahedralized model enable parallel, bounded and discontinuous surfaces

5.2 Review of Reconstruction Methods

Surface reconstruction from unorganized point data is a wide research topic. In the last few years, a number of different methods were investigated to solve this problem. One can divide the current techniques into three groups: *mesh independent* solutions, *explicit* approaches and methods using *implicit* functions.

5.2.1 Mesh Independent Point Set Rendering

With increasing performance of current graphics hardware, point set rendering methods are getting more and more popular. One common aim of point set rendering methods is high rendering performance and quality. Levoy and Whitted [1985] suggested to use points as basic graphic primitives instead of traditional primitives like triangles or polygons. Many point-based algorithms locally define a surface as the graph of a function. Every local estimation is based on a polynomial function that was computed over a projection plane. This can be done by the Moving Least Squares (MLS) approximation [Levin, 1998, 2003]. Dey and Sun [2005] extended the MLS method to noisy non-uniform sampled data sets and offer theoretical guarantee for the reconstruction. Alexa et al. [2001] supply a method where the density of points is dynamic, achieving high quality rendering. Fleishman et al. [2005] use a projection operator to reconstruct a piecewise smooth surface. Then, they expand the MLS method and take sharp features into account. Scheidegger et al. [2005] use the MLS approach to estimate the surface normal and curvature of a noisy point set. An advancing front algorithm triangulates the surface with adaptive triangle size. Although this method supports boundaries, it was developed for dense point sets. Adamson and Alexa [2003] worked on a simplified ray intersection procedure and applied their method to irregular point data. Because MLS-based solutions often introduce artifacts, Klein and Zachmann [2004] proposed a method based on a Voronoi diagram. Computing geodesic distances on the surface adapted from the Voronoi diagram ensures the robustness of their algorithm. Ohtake et al. [2005] reconstruct a surface by composing nonconforming overlapping surfels [Pfister et al., 2000]. Each surfel defines a local polynomial approximation of the resulting surface. Nevertheless, point set rendering methods do not yield good results if the input data is ill-conditioned; and this is often the case for scientific applications dealing with natural objects (e.g. geology).

5.2.2 Explicit Methods

Explicit methods directly create a surface representation from the input point set. Most of the time, the result is a Delaunay-triangulated surface. A widespread explicit method is the α-shapes proposed by Edelsbrunner and Mücke [1994]. A $3d$-complex is carved out by an α-ball until a so-called α-shape remains. That shape is used to reconstruct the surface. Many sequencing authors used the concepts of α-shapes like [Bajaj et al., 1995]. Bajaj et al. used α-shapes to compute a piecewise linear approximation of the domain to reconstruct. On this approximation, they compute a signed distance transform and incrementally build a tetrahedral decomposition that is used to compute the reconstruc-

tion consisting of smooth polynomial Bernstein-Bézier surface patches. Bernardini et al. [1999] used α-shapes together with a mesh simplification method. They recreated the surface on the approximated mesh with piecewise algebraic surface patches. The Power Crust algorithm of Amenta et al. [2001] also uses some concepts of α-shapes but its main algorithm is based on the medial axis transform that is computed in a first step. Then, they recreate the surface with an inverse transform. Amenta's algorithm offers a theoretical guarantee for dense data sets and homeomorphic surfaces. Mederos et al. [2005] extended the Power Crust method for noisy data sets. Their method honors the data precisely and thus the reconstruction results in a rough surface. Another group of explicit methods is the reconstruction of point data with spline patches. Assembling these patches to ensure high-order continuity is very difficult. Grimm and Hughes [1995] proposed a method to calculate B-Spline patches over a user-defined control mesh. The patches are sewed together using manifolds. Recently, Ying and Zorin [2004] investigated the idea of using manifolds to create C^∞ continuous surfaces. Kolluri et al. [2004] build a tetrahedral mesh on the domain of the scattered point cloud as in our method. Kolluri et al. decompose the volume with a huge number of tetrahedrons (several million) and use a spectral graph to classify the simplices into interior and exterior cells. The reconstruction is a triangulated surface where the interior and exterior cells intersect each other. This method always produces watertight manifold surfaces. In the frame of GOCAD research, Tertois and Mallet [2005] developed a method to fit single-valued surfaces to huge point clouds. This method is specialized on the reconstruction of single-valued geological interfaces like faults and horizons.

5.2.3 Implicit Methods

Implicit methods reconstruct a surface from an iso-value of an implicit function. These methods have several advantages compared to explicit methods. Implicit methods are more resistant to noise, fill holes automatically and support arbitrary topology. Hoppe et al. [1992] first introduced in their pioneering paper the problem of surface reconstruction using a signed distance field. Later Curless and Levoy [1996] published an algorithm to create a distance function over a regular volumetric tetrahedral mesh. The reconstructed surface is an iso-value surface of the distance function with a value zero. Ohtake et al. [2003] extended the distance field method to more than a million data points. The signed distance transform is computed by an adaptive octree space partitioning and local shape functions. Xie et al. [2004] used signed distance fields based on the method of Shepard [1968].

A second group of implicit methods uses Radial Basis Functions (RBFs), which are spherical-symmetry functions centered on a particular point. Savchenko et al. [1995] introduced in their pioneering publication the interpolation of an implicit function on a carrier object. They defined the solid as $\varphi(x,y,z) \geq 0$ and the surface as $\varphi(x,y,z) = 0$. Turk and O'Brien applied RBFs first for shape transformation [1999], and later on for shape modeling [2002]. They also suggested new methods to define the interior of the surface by defining either interior or exterior constraints. An interior constraint corresponds to a given positive value of the implicit function at a data point, while an exterior

constraint corresponds to a negative value. Morse et al. [2001] improved the RBF-based technique by using compactly sorted radial basis interpolants. They were able to apply the RBF method to several thousand constraints. [Carr et al., 2001] first used RBFs to resample large data sets using up to a half a million basis functions and later combined the RBF with smoothing convolution kernels [Carr et al., 2003]. A strong limitation of the methods using RBFs is that they can only reconstruct surfaces without noise and discontinuities without further processing.

The developed algorithm also uses the idea of defining bounding, interior and exterior constraints but it is based on a mathematical model that totally differs from the RBF model. The RBF methods install up to one basis function for each constraint, whereas the introduced method does not primarily depend on the number of constraints but on the magnitude of the tetrahedral mesh. The next sections describe this approach in detail.

5.3 Reconstruction Algorithm

The algorithm developed in this work can be broken down into three steps:

1. Creation of a tetrahedral mesh with adaptive resolution. Discontinuities are introduced into the tetrahedralized model, to later define discontinuities of the reconstructed surface and to prevent the connection of parallel surface patches (e.g. very thin salt bodies).

2. Interpolation of an implicit function $\varphi(x,y,z)$ over the tetrahedral mesh. The surface to be reconstructed is the implicit surface defined by $\varphi(x,y,z) = 0$.

3. Surface reconstruction from the implicit function and beautification of the triangulated mesh.

Figure 5.3 shows a flowchart of the reconstruction process. The following sections describe the three steps in detail.

5.3.1 Volume Tessellation

The first step of this reconstruction algorithm is the creation of a $3d$-simplicial complex that covers the object O to be reconstructed. The quality of this tetrahedral mesh is essential for the reconstruction result. Its generation can be described as follows.

Euclidean Distance Transform

Before performing the tessellation, a regular background grid carrying a $3d$-unsigned Euclidean Distance Transform (EDT) on a voxelized domain of the object O is computed. The resulting $3d$-distance map D contains the distance of any given point p to the closest point of the scattered point data.

$$D(p) = \min\{dist(p,q), q \in O\} \tag{5.1}$$

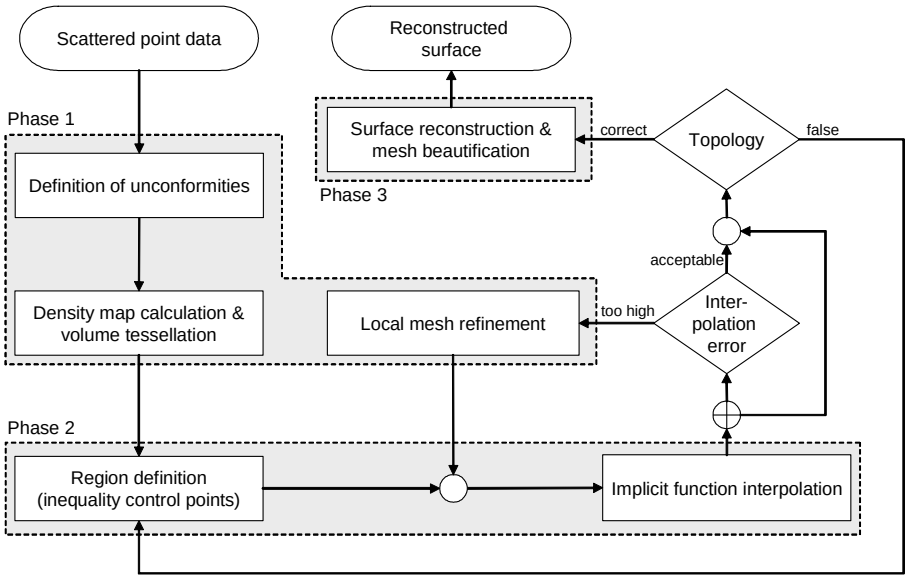


Figure 5.3: This flowchart illustrates the reconstruction process including quality control loops. A tetrahedral mesh covering the domain of the point data is created in the first phase. The second phase includes the definition of the regions divided by the reconstructed surface and the interpolation of the implicit function $\varphi(x,y,z)$. The surface is reconstructed from the implicit function and the triangular mesh is beautified in the final phase. To handle quality issues like interpolation or topological errors one can repeat selected operations of the first two phases.

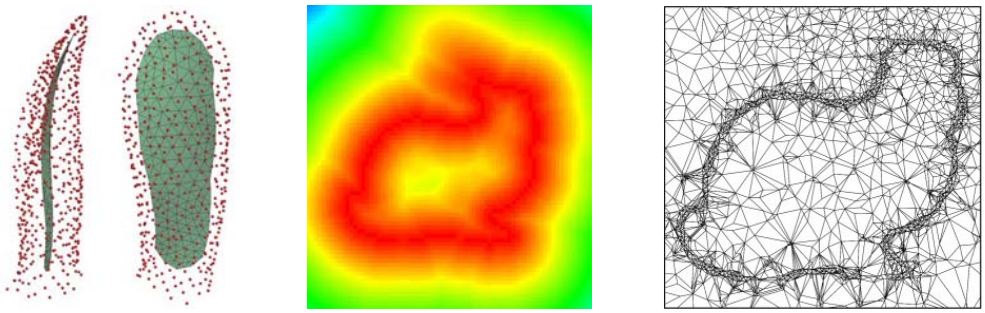
Where $dist(p,q)$ is the Euclidean distance between the point p of the voxelized background grid and a point q of O. Section 3.3.2 illustrated the computation of such an EDT.

Defining Discontinuities

A second preprocessing step is the definition of discontinuities. Discontinuities in the tessellated model create discontinuities in the implicit function. Figure 5.5(c) illustrates a discontinuous implicit function with highlighted discontinuities. Discontinuities are given surfaces that define a discontinuity in the complex where the nodes of the tetrahedral mesh will be dissociated. Surface boundaries and discontinuities that often occur in scientific applications can be modeled this way. Further, narrow regions like very thin salt bodies require a special treatment. The interior of two parallel surface patches bordering a narrow region of the object to be modeled is cut by a surface (see figure 5.4(a)). This cut or unconformity later prevents the interpolation algorithm from connecting these patches and producing undesirable handles. Once the density map and the discontinuities were defined, one can build the $3d$-complex around the input point set.

Volume Tessellation

By definition the $3d$-simplicial complex created for this algorithm is a tetrahedral mesh of the $3d$-Euclidean space, for which an intersection of two tetrahedrons is either equal to a common triangular facet, a common edge, a common vertex or is empty. Here the $3d$-



(a) (b) (c)

Figure 5.4: (a) shows an ear of the Stanford bunny that was cut by a surface. This cutting surface defines an unconformity of the $3d$-complex. (b) shows a cross-section through a distance map that is based on an Euclidean Distance Transform (EDT). The closer a point p is to the object O (input point data), the denser the tetrahedra tessellated mesh. (c) shows a cross-section through the tessellated model. The regions near the input data points are occupied by small tetrahedrons whereas the distant regions are covered by large tetrahedrons.

complex occupies the whole domain of object O. For the sake of simplicity, the interior of a surrounding cube is tessellated. The tetrahedral mesh is Delaunay constrained [George and Borouchakri, 1997]; that means that no node from another tetrahedron is located in the circumscribing sphere of each tetrahedron. The tessellation is performed using the methods introduced by Lepage [2003].

The previously computed distance map D is used as a density map for the generation of the tetrahedra tessellated model. The closer a point p is to the input data, the denser the tetrahedral mesh. As a consequence, the region around the points of the input data is populated with many very small tetrahedrons and the regions outside of the scope of interest are filled up with large tetrahedrons. The tetrahedrons that share a node, an edge or a face with an unconformity are disconnected from the corresponding neighbor on this contact. This disconnection allows a discontinuity of the implicit function $\varphi(x,y,z)$ that is interpolated in the next step over the tetrahedral mesh. If the interpolation error is too high (surface does not match input data closely enough) or topological ambiguities have to be resolved one can repeat certain operations of the first two phases of the reconstruction process to achieve the desired results (see also figure 5.3). Section 5.3.3 deals with quality control in more detail.

Implicit surfaces are $2d$-geometrical objects that are embedded in the $3d$-space. An implicit surface can be considered as an iso-value surface of the corresponding implicit function. If a point $(x_i,y_i,z_i) \in O$ lies on the surface, the value $\varphi(x_i,y_i,z_i)$ of the implicit function must be equal to zero [Bloomenthal and Wyvill, 1997]. Let us consider a point cloud, then the implicit function $\varphi(x,y,z) < 0$ defines one side and $\varphi(x,y,z) > 0$ the opposite side of the point cloud. The set of points (x,y,z) such that $\varphi(x,y,z) = 0$ represents a surface interpolating the given data set. The following sections show how to create such an implicit representation.

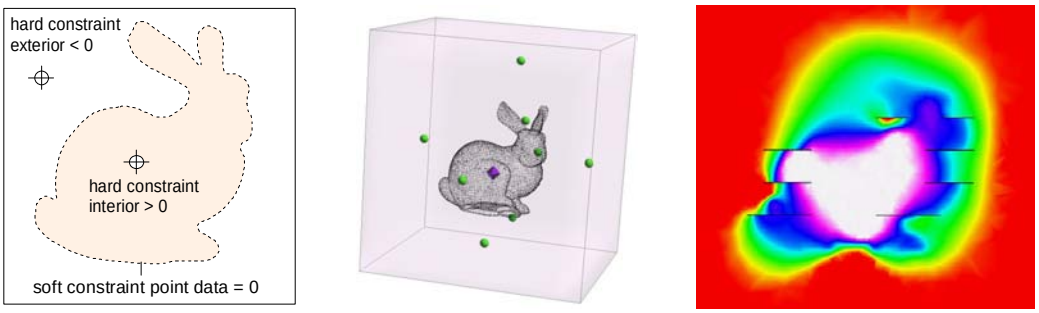


Figure 5.5: (a) illustrates how to define regions for the reconstruction algorithm. The input data points define soft constraints with the value zero. On the inside of object O, an inequality control point constrains the value of the implicit function to be positive. Other control points constrain this function to be negative on the outside of O. (b) shows the Stanford bunny point cloud (8,171 points) without discontinuities. Spheres are inequality control points that define the outside of the bunny and the diamond is an inequality control point defining the inside of the bunny. (c) shows a cross-section through the tetrahedra tessellated complex for the bunny with discontinuities shown in figure 5.2(a). The visualized property is the interpolated implicit function. Its discontinuities are clearly visible.

5.3.2 Implicit Function on a *3d*-Simplicial Complex

This section describes the second phase of the reconstruction workflow (see figure 5.3). In this phase, a set of constraints is defined to interpolate an implicit function on a tetrahedral mesh. The notion of a linear function on a tetrahedron and the interpolation method to create this function is presented in chapter 1.

DSI Constraints

The point set is embedded into a tetrahedra tessellated complex. Before an implicit function $\varphi(x,y,z)$ can interpolated it is necessary to define some constraints for the interpolation. The input data points bear soft constraints (control point constraints) with value zero. Furthermore, one has to define two regions, one on each side of the surface to be reconstructed. In the case of a closed surface, the convention that the implicit function has positive values on the inside and negative values on the outside of the surface is used. For the definition of the regions at least two additional points, one for each region (see figure 5.5(a) and 5.5(b)) are needed. These inequality control points constrain the value of the implicit function to be greater or less than a given value at this location. A roughness constraint (constant gradient) ensures that the solution to this interpolation problem is smooth. Control point and constant gradient constraints were already introduced in chapter 1. Inequality constraints are explained in the following.

Inequality Constraints (Region Definition) The inequality constraints define the two sides of the surface to be reconstructed. Therefore, at least one inequality control point on one side with the constraint $\varphi(x,y,z) \geq \varepsilon$, and at least one inequality control point on the other side of O with the constraint $\varphi(x,y,z) \leq -\varepsilon$, where ε is a small positive value (see

figure 5.5(a) and 5.5(b)) is defined. These inequality constraints are hard constraints and have to be honored in any case by the DSI method. To ensure this condition the Conjugate Gradient method was modified in the following way [Mallet, 2002].

The column matrix φ of size n is the current solution. According to the framework of optimization theory [Gill et al., 1982] the set $\mathscr{A}(\mathscr{C}^h|\varphi)$ of "active" hard constraints are defined:

- Greater than constraint $A_c^t \cdot \varphi \geq b_c$:

$$\begin{array}{lcll} A_c^t \cdot \varphi \geq b_c & \Rightarrow & c \notin \mathscr{A}(\mathscr{C}^h|\varphi) & \text{constraint is inactive} \\ A_c^t \cdot \varphi < b_c & \Rightarrow & c \in \mathscr{A}(\mathscr{C}^h|\varphi) & \text{constraint is active} \end{array} \tag{5.2}$$

- Less than constraint $A_c^t \cdot \varphi \leq b_c$: can easily derived from the "greater than" constraint by multiplying $A_c^t \cdot \varphi \geq b_c$ by $-1 \Rightarrow -A_c^t \cdot \varphi \leq -b_c$

A hard constraint is said to be inactive when it is honored. So φ is a valid solution of the minimization problem when the set of hard constraints $\mathscr{A}(\mathscr{C}^h|\varphi)$ is empty. The active set method proceeds as follows. Let us assume that φ^0 is an initial solution that honors all hard constraints. We are looking for a new point φ (respectively new solution) in the space $\mathbb{R}^n$ so that φ minimizes the violation criterion of equation (1.22). For this new solution we rebuild the set of active constraints $\mathscr{A}(\mathscr{C}^h|\varphi)$. To respect the hard constraints, we look for a point φ^* that is close to φ and where $\mathscr{A}(\mathscr{C}^h|\varphi)$ is empty. Every constraint $c \in \mathscr{C}^h$ can be interpreted as a hyperplane H_c defined by the equation $A_c^t \cdot X - b_c = 0$. The solution φ^* is the closest point in the $\mathbb{R}^n$ space from φ to H_c. A_c is orthogonal to H_c and the orthogonal projection φ^* of φ onto H_c can be computed as:

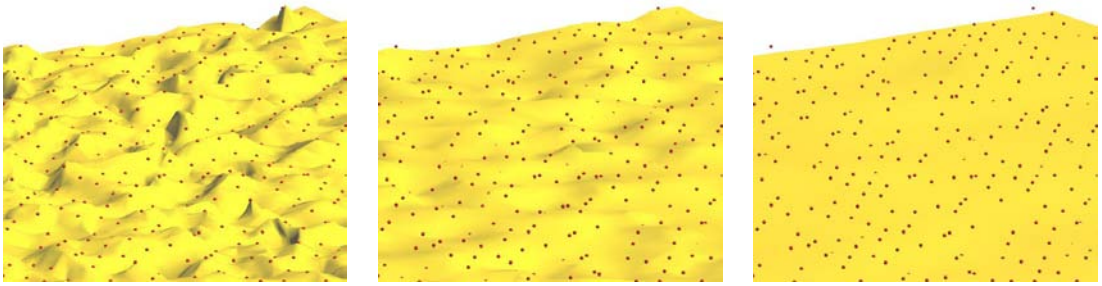
$$c \in \mathscr{A}(\mathscr{C}^h|\varphi) \rightarrow \left| \begin{array}{rcll} \varphi^* & = & \varphi - \mu \cdot A_c & \text{with:} \\ \mu & = & \frac{A_c^t \cdot \varphi - b_c}{\|A_c\|^2} & \end{array} \right. \tag{5.3}$$

The hard constraints are honored when the active set is empty. To empty an active set an iterative method described by listing (5.1) is used.

Listing 5.1: Pseudo code for emptying an active set

```
while A(C^h|φ) is not empty do
  choose a constraint c ∈ A(C^h|φ)
  compute μ ← (A_c^t · φ − b_c) / ||A_c||^2
  update φ ← φ − μ · A_c
  update A(C^h|φ)
end while
```

As all constraints $c \in \mathscr{C}^h$ have to be consistent, listing 5.1 converges. To honor the hard constraints for the solution the algorithm from listing 5.1 is inserted into the Conjugate Gradient method (see listing 5.2).



(a) $\varpi_c^{roughness}/\varpi_c^{property} = 0.01$ (b) $\varpi_c^{roughness}/\varpi_c^{property} = 1$ (c) $\varpi_c^{roughness}/\varpi_c^{property} = 100$

Figure 5.6: These figures show a cutaway of a reconstruction of a noisy point set where the weight of roughness and property constraints was varied. (a) shows a flexible surface that honors all input data points. (b) has the same weight for roughness and property constraints. Not all constraints are honored. (c) shows a reconstruction with a high stiffness that interpolates the input data.

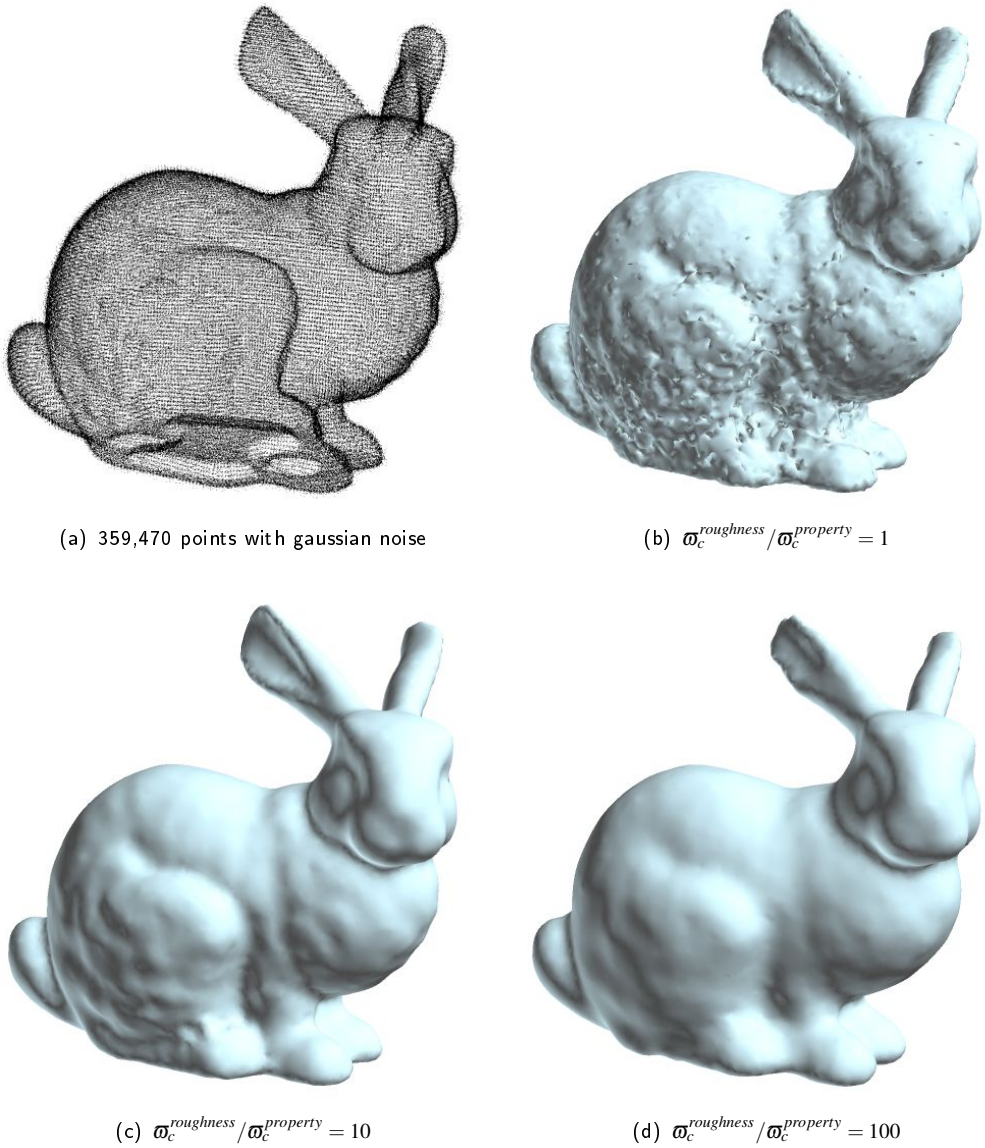
Listing 5.2: Pseudo code for the modified Conjugate Gradient method

$\varphi \leftarrow \varphi_0$ *// initial solution*
$G \leftarrow \mathbf{A} \cdot \varphi - \mathbf{b}$ *// gradient of* φ
$D \leftarrow -G$ *// search direction*
while convergence **not** reached **do**
  $\lambda \leftarrow \frac{-G^t \cdot D}{D^t \cdot \mathbf{A} \cdot D}$
  $\varphi \leftarrow \varphi + \lambda \cdot D$
  $G \leftarrow \mathbf{A} \cdot \varphi - \mathbf{b}$
  $\beta \leftarrow \frac{G^t \cdot \mathbf{A} \cdot D}{D^t \cdot \mathbf{A} \cdot D}$
  $D \leftarrow \beta \cdot D - G$
  empty the active set $\mathscr{A}(\mathscr{C}^h|\varphi)$
end while

Using this modified Conjugate Gradient method, the implicit function $\varphi(x,y,z)$ at each node of the tetrahedral mesh is computed. As figure 5.5(c) shows, function $\varphi(x,y,z)$ is then linearly interpolated inside each tetrahedron. The implementation of the inequality constraints within the MxDSI framework was performed by Anne-Laure Tertois [Muron et al., 2005].

Handling Noise

For noisy data sets it is not always desirable to honor all input points in order to obtain a smooth reconstruction. DSI honors the constraints in a least squares sense. By weighting the different constraints (see equation 1.19) such as property control points and roughness criterion, this method is capable of controlling the smoothness of the reconstructed surface. Figure 5.6 illustrates reconstructions for different roughness/property constraint ratios. The more important the roughness constraint is compared to the property constraint, the smoother the reconstruction will be. The weight of the roughness constraint controls the stiffness of the reconstructed surface. 5.7(a) shows a point cloud of the Stanford bunny with 359,470 points with gaussian noise. The reconstructions were performed



(a) 359,470 points with gaussian noise

(b) $\varpi_c^{roughness}/\varpi_c^{property} = 1$

(c) $\varpi_c^{roughness}/\varpi_c^{property} = 10$

(d) $\varpi_c^{roughness}/\varpi_c^{property} = 100$

Figure 5.7: These figures show the reconstruction from a noisy point set of the Stanford bunny (a). Too low (b) or too high (d) roughness property constraint ratios lead to artifacts or melting of the reconstruction; (c) shows the reconstruction of an optimal value for the roughness property constraint ratio without artifacts and sufficient details.

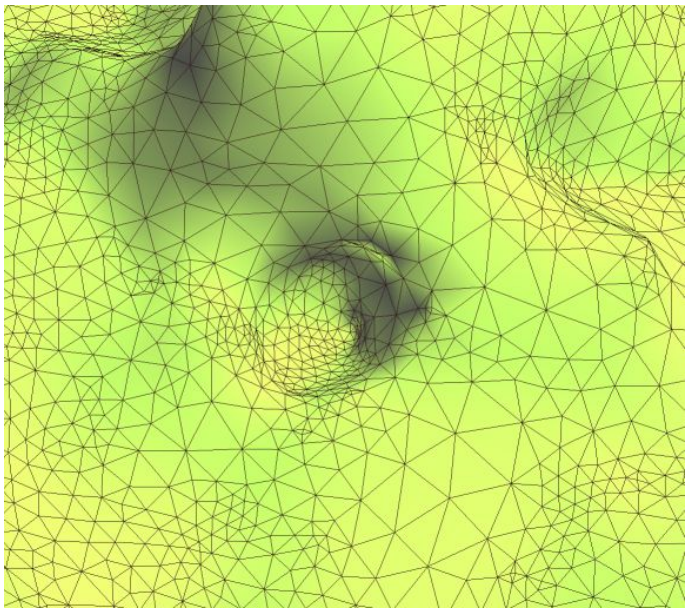


Figure 5.8: This figure shows a cutaway from a reconstructed salt-top surface. Regions with a high curvature are represented by small triangles whereas flat regions can be reconstructed with large triangles. The adaptive triangle size is based on a densification of the tetrahedral mesh in regions with a high interpolation error.

with different roughness property constraint ratios. For a low ratio the reconstruction shows a lot of artifacts (see figure 5.7(b)), whereas a too high ratio melts the reconstruction (see figure 5.7(d)). An optimal ratio results in no artifacts but respects details (see figure 5.7(c)).

Once the implicit function $\varphi(x,y,z)$ is built, an optional quality control loop can be run, and finally the surface is reconstructed from the implicit function.

5.3.3 Quality Control

The reconstruction workflow proposed in figure 5.3 is an iterative approach where interpolation errors can be eliminated by gradually refining the tetrahedral mesh and hereby improving the resolution of the interpolated implicit function $\varphi(x,y,z)$. Micro-topological errors like bubbles or capillary tubes can be detected and eliminated automatically. Macro-topological ambiguities – that cannot be resolved by the reconstruction algorithm itself – can be resolved by defining additional region constraints (inequality control points).

Interpolation Error

This method also defines a local incremental quality control. This can be useful for finely detailed regions of the reconstructed surface where the initial resolution of the tessellation

is insufficient. For each point of the input data, an interpolation error $|e|$ is computed.

$$|e| = \frac{|\Delta\varphi|}{\|\nabla\varphi\|} \tag{5.4}$$

$|\Delta\varphi|$ is the difference of the value of the data point and the value of the implicit function $\varphi(x,y,z)$ at that location. We define the input data with the value zero, so $|\Delta\varphi| = |\varphi(x,y,z)|$. Further, $\|\nabla\varphi\|$ is the Euclidean norm of the gradient of $\varphi(x,y,z)$. It can be shown that $|e|$ is an approximated distance of the input data point to the implicit surface defined by $\varphi(x,y,z) = 0$. The control points that have an error that exceeds a certain threshold are used to refine the tetrahedral mesh locally (see chapter 4). After mesh refinement, the interpolation is run again and the implicit function respects the control node constraints of the input data – due to the higher resolution of the tetrahedral mesh – more precisely. The reconstructed surface is characterized by small triangles at regions with high curvature where the initial mesh was densified and larger triangles in planar regions where the initial tetrahedral mesh showed sufficient resolution (see figure 5.8).

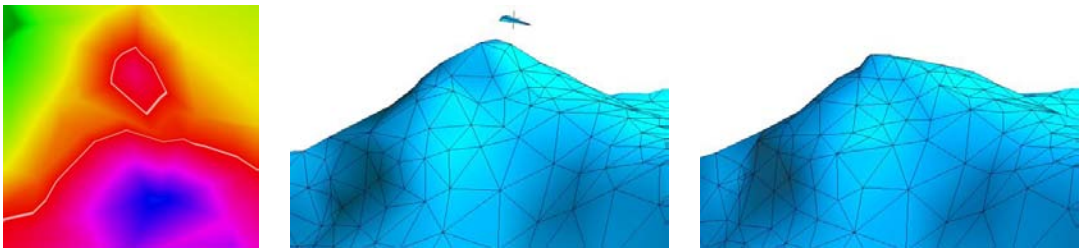
Local Extrema of $\varphi(x,y,z)$

Due to noise in input data and tetrahedrons with bad quality the implicit function $\varphi(x,y,z)$ can take local extrema. Some of these extrema can lead to additional interfaces (bubbles or capillary tubes) of the iso-value surface representing the reconstruction. For the detection of these artifacts two methods were applied:

Direct Method Local extrema of $\varphi(x,y,z)$ must take negative values on the positive defined side of the surface and positive values on the negative defined side of the surface to introduce an artifact. Nodes fulfilling this condition can be identified by checking the neighborhood for local extrema. This method can also be used to detect other artifacts like sharp features or capillary tubes.

Indirect Method The indirect method works on the reconstructed surface and not on the implicit function. The reconstruction is analyzed for closed surface parts. Closed parts that fall under a certain size criterion are qualified as artifacts. The advantage of this method is that it is extremely fast but limited to bubbles.

Once artifacts like bubbles or capillary tubes were detected, the involved nodes of the tetrahedral mesh are impinged with an additional constant gradient constraint with a very high weight compared to the standard roughness. This additional impact on the local roughness of the implicit function increases the stiffness of the iso-value surface and hereby decreases its capability to form artifacts. Figure 5.9(b) shows a reconstruction with a small undesired bubble. The automatic detection marked this with a cross. After the installation of a locally strong impact of the roughness constraint the artifact disappeared (see figure 5.9(c)). Due to the stronger weight of the roughness constraint the surface has a higher stiffness and flattens on the peak of the summit.

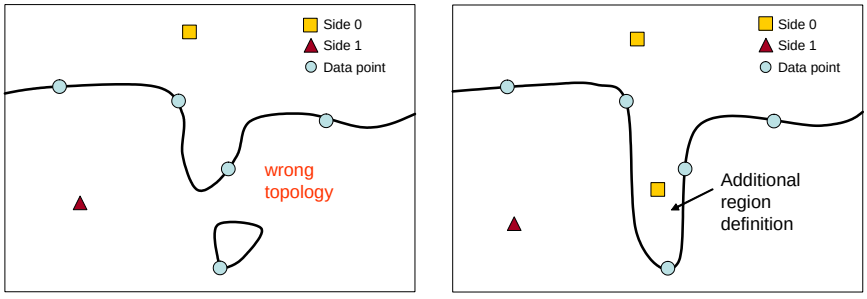


(a) (b) (c)

Figure 5.9: (a) shows from a cross-section through a tetrahedral mesh. The Implicit function shows a bubble-like artifact. The white iso-contour represents the trace of the reconstruction. (b) shows a bubble in the reconstruction result; (c) shows the same reconstruction with a strong impact of the roughness constraint in the region of the former bubble. The stiffness of the reconstructed surface is increased and the summit is flattened.

Macro-topological Ambiguities and Shape Modeling

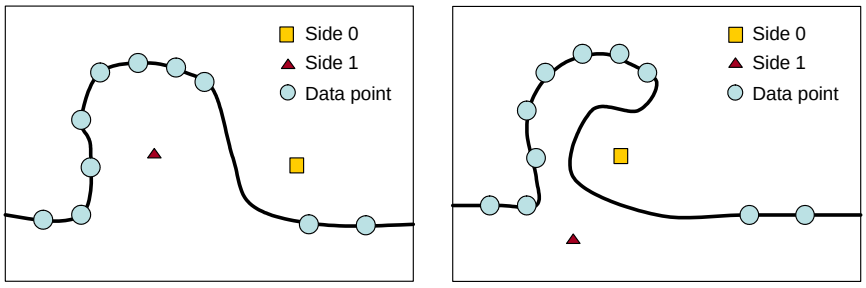
Input data from scientific applications are often noisy or sparse. Such data can result in topological ambiguities that cannot be resolved without additional information. In this section we describe a method to resolve topological ambiguities by introducing additional information into the reconstruction process. Figure 5.10 shows a configuration where the input data define a steep valley. Due to the sparse density of the input data the interpolation might cut off the valley and create an additional bubble to optimize the smoothness of the implicit function (see figure 5.10(a)). This case of topological uncertainty can be eliminated by manually inserting an additional inequality constraint to deform the shape of the reconstructed surface (see figure 5.10(b)).



(a) (b)

Figure 5.10: These figures show a sparse point set that defines a steep valley. The valley was cut off and a bubble was formed on the separated data point (a). By introducing an additional inequality control point for region definition, this topologically false ambiguity can be eliminated (b).

Figure 5.11 shows the contour of a reconstructed surface. The input data points have a gap. By default the reconstruction method connects the missing data as smoothly as possible (see figure 5.11(a)). If this smooth interpolation does not represent the real shape of the surface one can again use additional inequality control points or displace existing



(a) (b)

Figure 5.11: These figures show a point set with missing data. By default the reconstruction method interpolates the missing data as smoothly as possible (a). The shape of the region with missing data can be modeled by inserting additional inequality constraints or by displacing existing control points that define the regions of the reconstructed surface (b).

control points used for region definition to deform and model the reconstructed surface (see figure 5.11(b)).

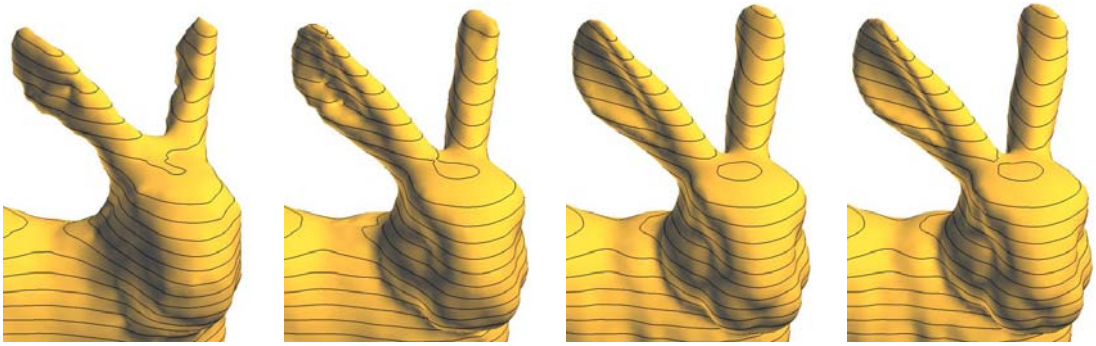
5.3.4 Surface Reconstruction

Once the implicit function $\varphi(x,y,z)$ is computed on the tetrahedral mesh, the implicit surface defined by $\varphi(x,y,z) = 0$ is reconstructed. For the extraction of the surface an adaption of the Marching Cubes (MC) [Lorensen and Cline, 1987] algorithm, called the Marching Tetrahedrons (MT) is used. As we are already working on a tetrahedralized domain it makes sense to use the MT and, in addition, the MT resolves the ambiguous cases of the MC. After reconstruction, the triangulated surface can be beautified for equilaterality of the triangles, using a Laplace filter or a $2d$-DSI algorithm [Mallet, 2002].

5.3.5 Performance Analysis

For an estimation of the performance[5] and stability of the developed method, the Stanford bunny with different resolutions was reconstructed (see figure 5.12). The numerical results are summarized in table (5.1). The complexity of the single stages of this algorithm depends on different parameters. The installation of the constraints (input data and region definition) depends linearly on the number of constraints $\mathscr{C}$ with $\mathscr{O}(|\mathscr{C}|)$. The installation of the roughness constraint depends linearly on the number of nodes of the tetrahedral mesh $\mathscr{O}(|\Omega|)$. The overall complexity of the constraint installation is thus $\mathscr{O}(|\Omega| + |\mathscr{C}|)$. In table (5.1), one can observe that the time required to install the constraints strongly depends on the number of nodes involved in the roughness constraint. Installation time only increases slightly when the number of data points increases by three orders of magnitude. The Conjugate Gradient method converges in at most n-steps where n is the number of unknown variables. In this case, the unknowns are the values of the implicit function on

[5] All benchmarks were performed on a mobile Pentium 4 CPU with 3.06 GHz, 1 GByte of RAM and a NVidia GeForce FX Go5200 with 64 MByte memory



(a) 453 points (b) 1,889 points (c) 8,171 points (d) 35,947 points

Figure 5.12: This figure shows cutaways of the reconstructed Stanford bunny. The reconstructions show the stability of the developed algorithm with different resolutions. Note that the genus of the parts of the surface corresponding to the ears is correct (no handles). The number of points refer to the whole bunny. The black contours are horizontal level curves. Data by courtesy of Stanford $3d$-Scanning Repository.

the nodes of the tetrahedral mesh and thus the complexity is $\mathcal{O}(|\Omega|)$. To respect the active set $\mathcal{A}(\mathcal{C}^h|\varphi)$ while solving the linear system, a worst case complexity of $\mathcal{O}(|\Omega| * |\mathcal{C}^h|)$ is postulated.

Number of input points	Constraint installation	Triangles	Triangles beautified
453	97 sec	80,543	34,801
1,889	98 sec	93,362	43,306
8,171	106 sec	95,624	46,782
35,947	117 sec	96,594	48,134
359,470	220 sec	110,070	43,118
Convergence Conjugate Gradient ≈120 sec			
Marching Tetrahedrons ≈4 sec			

Table 5.1: Reconstruction results for the Stanford bunny with different resolutions. The $3d$-complex consists of 197,776 tetrahedrons built on 32,878 nodes. The table presents the resolution of the point sets, the time needed for the installation of the constraints (computation time needed to initialize the DSI linear system) and the number of triangles of the reconstructed surfaces before and after the beautification.

It must be pointed out that this method does not primarily depend on the number of constraints (input data points) but on the number of nodes of the tetrahedral mesh. Thus, the convergence of the Conjugate Gradient method for the four different resolutions of the Stanford bunny is reached in roughly the same computation time (see figure 5.13). In the same way, the number of triangles of the reconstructed surface is correlated to the resolution of the tessellation in regions near the input data. The same tetrahedral mesh was used to reconstruct the different input data sets and consequently the Marching Tetrahedrons performed in the same time. Optional post-processing can be applied to beautify the extracted triangulated surface, reducing the number of its triangles by one half.

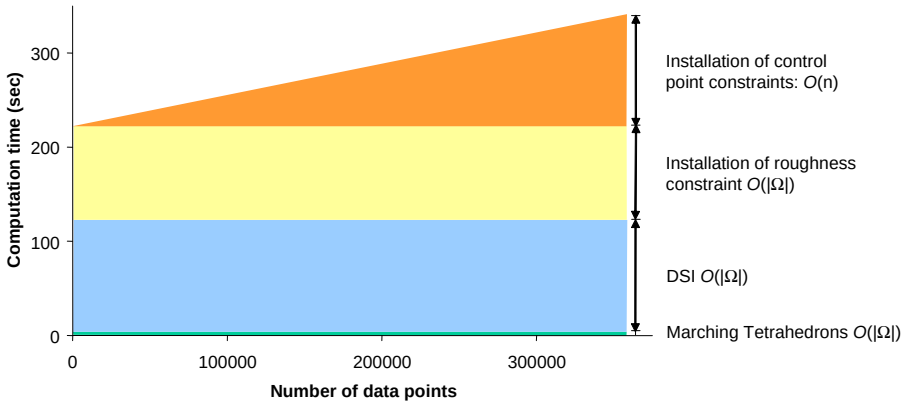


Figure 5.13: This illustration shows a schematic composition of the accumulated computation time for surface reconstruction. The overall performance is dominated by the number of nodes of the tetrahedral mesh and not by the number of input data points. Illustration is based on the data of table 5.1.

5.4 Results

The presented method was developed with a special focus on scientific data especially for the reconstruction of geological interfaces like complex horizons and salt domes. Figure 5.1 shows a salt-top surface with a single salt dome. This surface shows several regions with high curvature and non-singularity in Z-direction (overhangs). The input data were relatively noisy because they were assembled by independent successive interpreter picks. Figure 5.14 shows a magnified view of the salt dome itself. The non-singularity in Z-direction was reconstructed correctly without building bridges or wrong topological handles. The red balls are the input data and the reconstructed surface is well tessellated.

Iterative refinement of the tetrahedral mesh enables high resolution at complex regions that are represented with small triangles whereas large triangles represent flat regions of the reconstructed surface. Figure 5.16 shows a point cloud of a salt-top surface picked from seismic data and its reconstruction.

This method was developed on the Stanford bunny because its ears are very similar to geological salt bodies with narrow, almost parallel surface patches (see figure 5.15). Figure 5.17 shows a salt dome with this characteristic. From the cross-section through this salt dome a magnified view of narrow surface patches demonstrates the topological robustness of this method (see figure 5.17(b)).

Figure 5.2 illustrates two models with discontinuities. In geological applications these discontinuities are defined by fault surfaces. These surfaces are obtained by seismic interpretation or well data. The fault surfaces can be considered as given input parameters for the reconstruction of geological interfaces like horizons or salt domes. Figure 5.2(b) shows a reconstruction of a horizon with several discontinuities introduced by fault surfaces. The fault surfaces are outlined in red.

5.5 Summary

This first chapter dealing with implicit modeling of tetrahedral meshes introduced a surface reconstruction method with outstanding features. Therefore, starting with a point cloud acting as control point constraints that was embedded into a tetrahedralized domain and some few region definitions, an implicit function is interpolated on the tetrahedral mesh. The precision can be controlled by iterative mesh refinement and topological ambiguities can be resolved by additional region definitions. Further, implicit representation fills holes in the input data automatically but unconformities are respected by discontinuities in the simplicial complex. For noisy data sets a roughness criterion controls the smoothness of the reconstructed surface. The complexity of this reconstruction method does not primarily depend on the number of input points but on the number of nodes of the tetrahedral mesh. The developed algorithm was tested on complex real world models with up to 400,000 data points. The reconstruction of the surface was performed on standard consumer hardware in a couple of minutes. The reconstruction of non-watertight models and surfaces with boundaries is an outstanding feature for scientific applications. The following chapter introduces an algorithm to perform interactively rapid deformation of implicitly defined shapes.

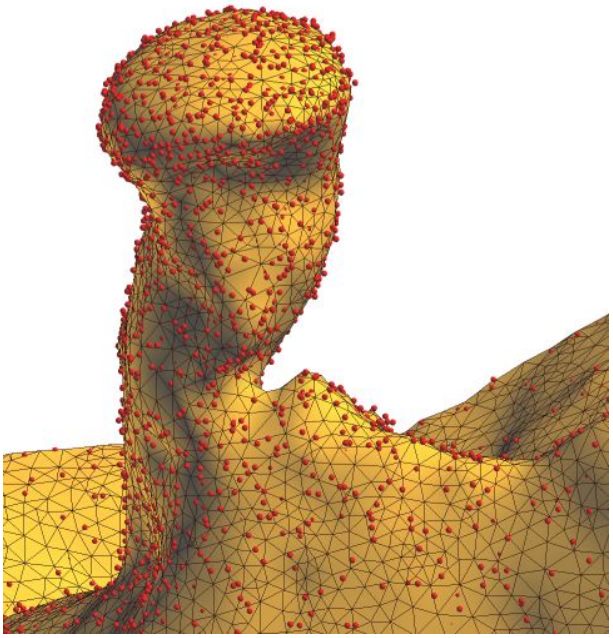


Figure 5.14: This figure shows a magnified view of the salt dome from figure 5.1. The trench (non-singularity in Z-direction) was correctly reconstructed without creating connections or wrong topological handles between the surface patches. The red balls represent the input data points. Data by courtesy of Shell.

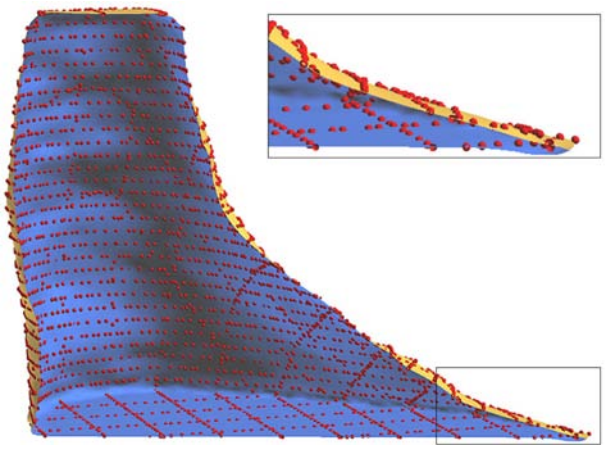
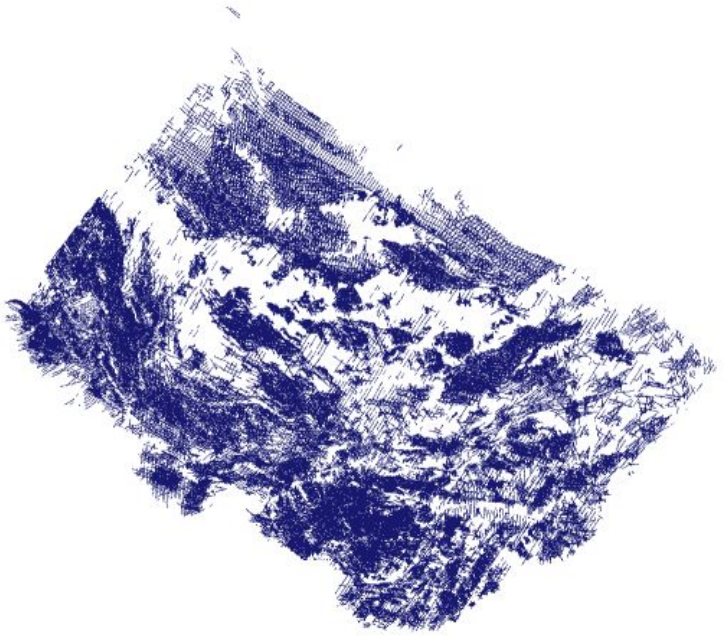
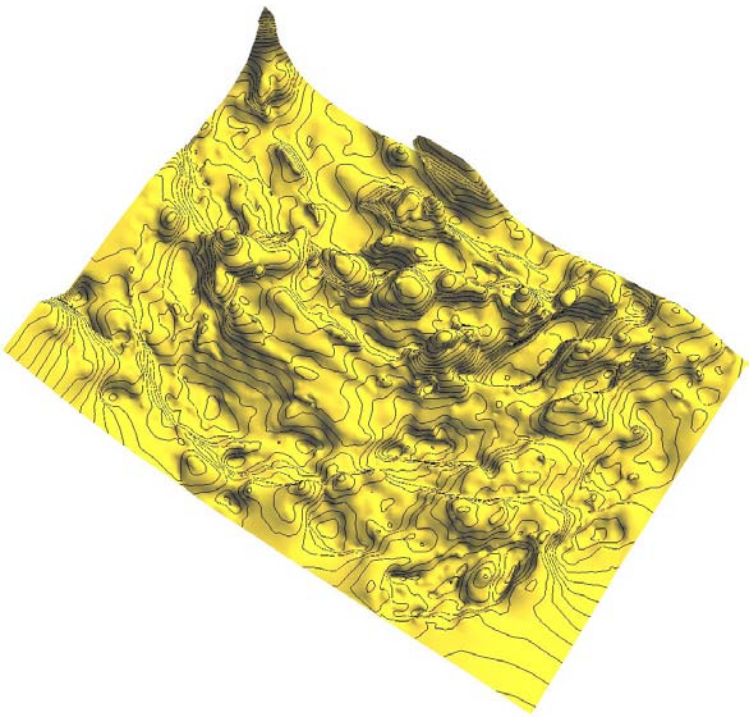


Figure 5.15: This figure shows a cross-section though a reconstructed salt-body. The right branch (magnification) is very thin and the top and the bottom salt-surfaces are almost parallel. The input data points are highlighted as red spheres. Data by courtesy of Shell.

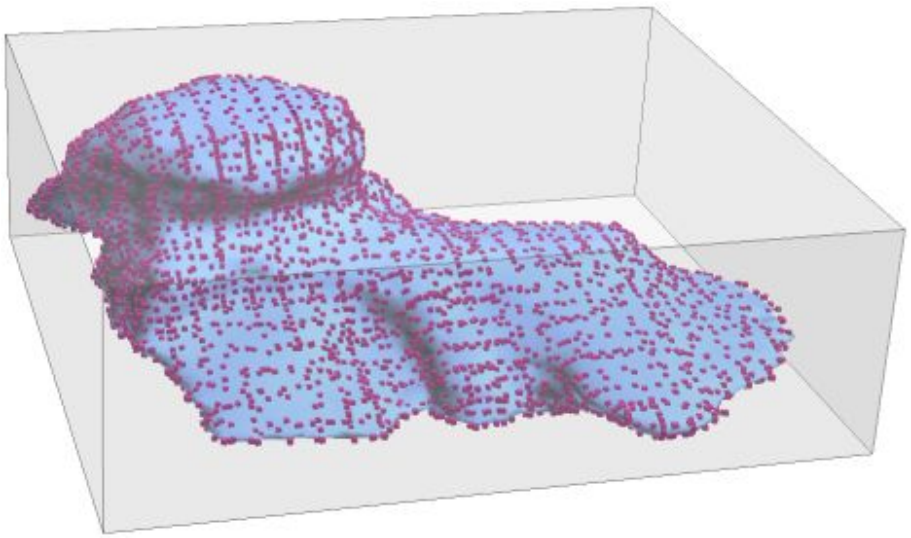


(a)

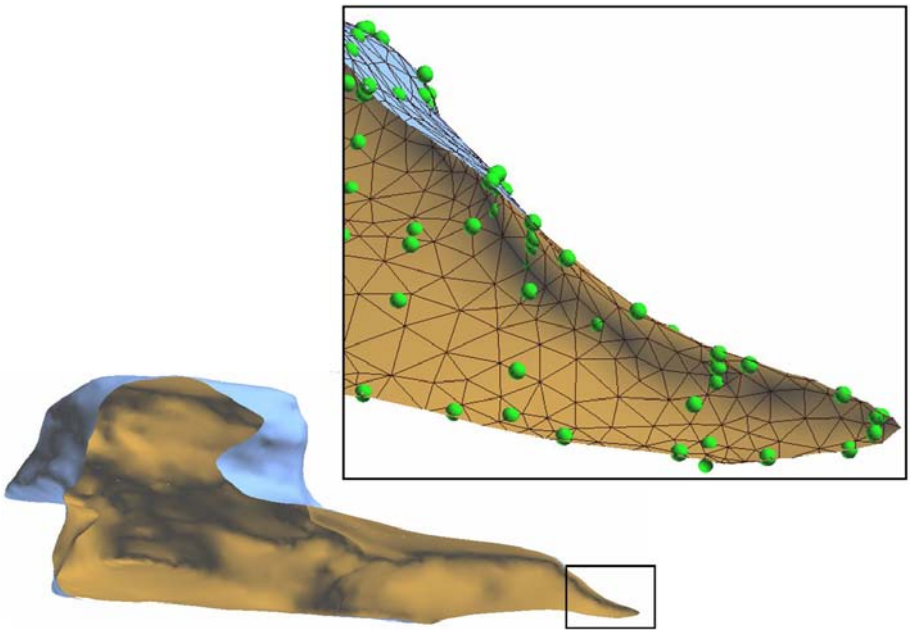


(b)

Figure 5.16: (a) shows 377,384 points picked from seismic data. This point set shows several large gaps and steep edges. (b) shows the reconstructed surfaces of (a) with 85,855 triangles. The implicit function was defined on a tetrahedral mesh with 95,073 tetrahedrons (15,625 nodes). This tetrahedral mesh was iteratively refined with 13 levels of Delaunay densification. The overall process took about half an hour of computation. Data by courtesy of Shell



(a)



(b)

Figure 5.17: (a) shows a salt dome (32,029 triangles) reconstructed from 4,384 points with its input data points. (b) shows a cross-section through the same salt dome. The region with very close parallel surface patches is magnified. The magnification also shows the input data points and the triangulation of the surface. The reconstruction was performed on a tetrahedra tessellated volume (141,863 cells) in 81 seconds. Data by courtesy of Unocal.

Chapter 6

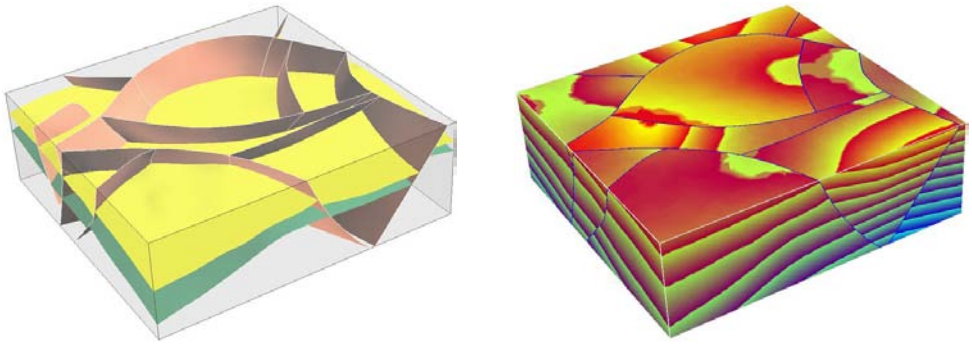
Deformation of Implicitly Defined Shapes

Contents

6.1 Introduction

Computer Aided Design (CAD) plays an outstanding role for industrial modeling, design and development. Today one can find three classes of mathematical model representation: *parametric*, *point sets* and *implicit*. Each category has its special field of application. Parametric representation like splines or triangle surfaces is widely used for classical CAD construction applications. As the fragment size of modern GPUs is getting smaller than the point size of dense data sets, the point set techniques are getting more and more popular. Implicit methods enable efficient computations of intersections and penetration of solids (see section 3.3.4) and are widely used to interpolate physical measurements on $3d$-meshes. Today, $3d$-representation is state-of-the-art and implies its own problems like creating and modeling the $3d$-model. The previous chapter dealt with the creation of complex implicit models on tetrahedral meshes. This chapter introduces a method to edit implicit models in real-time with immediate user feedback.

Model editing is a key technology for creating a precise representation of the geological subsurface. Geophysical information like seismic or well logs are used to build a structural model of the subsurface (see figure 6.1(a)). A structural model consists of faults, horizons and their topological relations. Such a model is always just an approximation of the subsurface. New insights, for example additional well data, may request a model



(a) Structural model

(b) Tetrahedral mesh with GeoChron time parametrization

Figure 6.1: (a) shows a structural model (fault network and reference horizons). (b) shows a tetrahedral mesh created from the structural model of (a). The property is a GeoChron time parametrization computed from the reference horizons. The nodes of the mesh were dissociated at the faults and their traces are highlighted with black outlines.

update. To avoid a rebuild of the model from scratch, modeling techniques are needed to change the shape and the location of faults and horizons in order to adjust and fine-tune the structural model. The displacement of faults – inside a tetrahedral mesh – was presented by Tertois et al. [2005]. This chapter introduces a method to deform horizons interactively respecting fault contacts and topological relations to other geological interfaces. Many approaches for model editing work directly on the explicit representation of boundary surfaces. These solutions have to deal with a lot of topological difficulties in order to maintain the consistency of the model (see figure 6.2).

The introduced method is based on implicit shape transformation. The model is represented by an unstructured tetrahedral tessellated volume. The faults are discontinuities in the graph of the tetrahedral mesh and the horizons can be extracted from a tri-variate geological time function φ defined on the nodes of this mesh (see figure 6.1(b)). This implicit function – based on the GeoChron parametrization (see section 1.2) – is a function where discontinuities are only introduced by unconformities like faults or salt domes. Reference horizons obtained by seismic or well data are iso-value surfaces $\mathscr{S}$ that take a distinct value of the function φ. Editing the implicit function deforms the iso-value surfaces of the regions that were affected by the modifications. The technical realization of the editing method is based on the displacement of property control points of φ and running a re-interpolation of the time function on the model using the Discrete Smooth Interpolation (DSI) method (see section 1.1.2). User feedback is obtained by re-computing $\mathscr{S}^*$ from the modified function φ^*. The algorithm reaches interactive frame rates for immediate user feedback.

6.2 Review of Deformation Methods

Shape deformations are a basic tool for animation and CAD applications. This section gives a short introduction on recent research topics of shape transformation techniques.

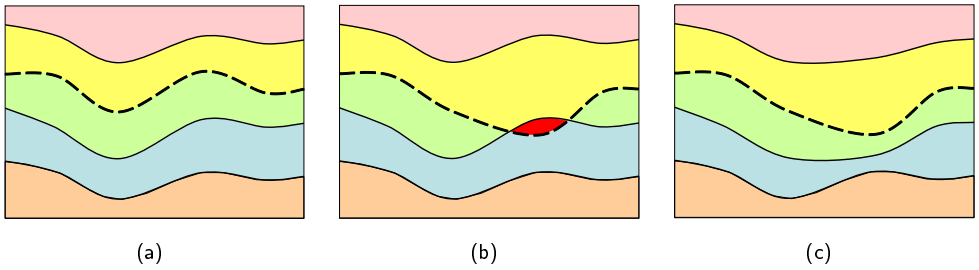


Figure 6.2: This series of figures shows a schematic view of a cross-section through a geological model with several horizons. (a) is the initial model. In figures (b) and (c) the dashed horizon was deformed. If the horizons are modeled as parametric surfaces without additional topological constraints the horizons can intersect each other and lead to inconsistent geological models as shown in figure (b). If the horizons are level sets of a continuous tri-variate implicit function, intersections cannot occur. Further, adjacent horizons follow the deformation (c).

The different techniques are divided into three groups: *Free-Form Deformations*, *editing of sealed geological models* and *implicit shape deformation* methods.

6.2.1 Free-Form Deformation

Free-Form Deformation (FFD) is a classic approach for surface editing. Sederberg and Parry [1986] displace the control points of tri-variate Bernstein polynomials to deform a parametric surface. Coquillart [1990] extended the method of Sederberg and Parry [1986] and introduced the Extended Free-Form Deformation (EFFD). Coquillart transformes a surface by adding bumps and bending the surface along user-defined curves. Chua and Neumann [2000] improved EFFD by integrating this method into the OpenGL API [Segal and Akeley, 2003]. Borrel and Rappoport [1994] introduced the Simple Constrained Deformation (SCoDef). B-Spline basis functions are centered at user-defined control points and their region of influence is determined by a radius. A displacement of a point causes a deformation by blending the affected basis functions. Schein and Elber [2004] extended FFD to discontinuous deformations (DFFD). They define a deformation function and the deformation is achieved by deforming the control volume of the deformation function. The unconformities are respected by discontinuities in the deformation function together with a model split algorithm. [Grosse, 2002] extended the FFD methods with geological constraints, creating a hexahedral control grid around a set of surfaces. Deformation is performed by displacing nodes of the control grids and the geological constraints are formulated as DSI constraints.

6.2.2 Sealed Geological *3d*-Models

Caumon et al. [2003, 2004] introduced methods to interactively edit geological models defined by a boundary representation (b-rep) [Weiler, 1988; Lepage, 2003]. This hierarchical model consists of fixed surfaces like faults or salt-top surfaces and deformable surfaces like horizons. Their method honors two constraints: a border of a deformable

surface can only slide along a defined surface and deformable surfaces like geological horizons may not intersect each other. The calculations are restricted to a region of influence and editing is performed in real-time.

6.2.3 Implicit Shape Transformation

Implicit representations of surfaces and bodies are widely used in CAD applications. Bloomenthal and Wyvill [1990] deformed implicitly defined shapes by modifying skeletons. A skeleton consists of points, splines and/or polygons and such a skeletal element is associated with a locally defined implicit function. Witkin and Heckbert [1994] introduced particles to control the shape of implicit surfaces. Control points fix the surface to certain regions and floaters deform the implicitly defined shape by its displacement. Turk and O'Brien [1999] transform shapes by interpolating n-dimensional implicit functions using thin plate interpolation in the $n+1$-dimensional space. Karpenko et al. [2002] extended this method with free-form modeling operations. In a later paper, Turk and O'Brien [2002] extended the idea of using particles to deform implicitly defined shapes. Additionally, they introduce the notion of constraints defining the two sides of an orientable surface. The interpolation is based on Radial Basis Functions (RBFs). Botsch and Kobbelt [2005] also use RBFs for shape editing. Their main focus lies on an incremental least squares solver to overcome the numerical complexity of dense linear systems introduced by tri-harmonic RBFs. Chen et al. [2003] propose a volumetric method together with transfer functions. Transfer functions are used to map different domains like a color-map and a physical property and are widely used in scientific visualization. Chen et al. use transfer functions to map locations in the Euclidean space and deform the initial model. Nealen et al. [2005] introduced a sketch-based editing method for surface meshes. The user can sketch a curve that is used as feature line. The deformation is performed honoring constraints on the normals and curvature to preserve details. Linear modeling constraints define the mesh modifications.

In addition, point sets are more and more used as graphic primitives [Alexa et al., 2001]. Of course shape transformations can be directly applied on these point set surfaces [Yoshizawa et al., 2002; Pauly et al., 2003]. In the frame of this work, we use a volumetric model. Thus, these methods cannot be applied and are not further pursued.

6.3 Interactive Editing of GeoChron Models

The method developed in the frame of this work is based on implicit functions defined on simplicial $3d$-complexes. The implicit function φ is defined on the nodes of the tetrahedral mesh. The shape editing is performed on an iso-value surface $\mathscr{S}_\phi$ where φ takes a constant value ϕ so that $\varphi - \phi = 0$. The basic idea of this method is to displace control points and to re-interpolate a deformed implicit function φ^* honoring the displaced control points. This idea is similar to Turk and O'Brien [2002] but uses a totally dif-

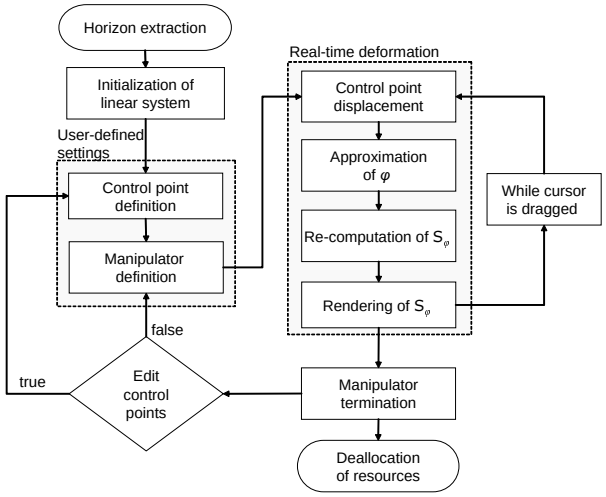


Figure 6.3: This figure represents the workflow to edit an implicit function φ defined on a tetrahedral mesh. In the preprocessing phase an initial linear system is solved. Control points on an extracted iso-value surface $\mathscr{S}$ define the modeling of φ. The displacement of a control point triggers the re-computation of φ. The deformed iso-value surface $\mathscr{S}$ is extracted and rendered. Control points can be added and removed before each manipulation.

ferent mathematical method. In this section an overview of the underlying workflow is introduced. Figure 6.3 shows a flowchart describing the model editing process:

Horizon extraction Before the implicit function φ can be edited one has to extract an iso-value surface $\mathscr{S}$. This extracted iso-value surface is the target of deformation. But one has to keep in mind that the deformed surface $\mathscr{S}^*$ is just an iso-contour of an edited implicit function φ^*.

Initialization of a linear system The initial solution of φ is obtained by a tri-variate interpolation of reference input data. This interpolation calculation is performed as a preprocessing step. The system of linear equations and its solution are needed for the later real-time deformation (see also section 6.3.1).

Control point definition The deformation of the iso-value surface is defined by a displacement of control points. The control points are created on the extracted iso-value surface $\mathscr{S}$. Control points that are not displaced in the later modeling process fix the values of φ in that location. Displaced control points edit φ according to their new locations. Control points can be added and deleted between each model manipulation. The displacement of a control point is defined by a manipulator. Additional control points are reference horizons. The weight of the different types of control points is user-defined.

Manipulator definition The displacement of a control point is performed interactively by dragging and dropping a control point with a cursor. Picking with a mouse on a $2d$-screen to move a point in three dimensions always implies the backward transformation from screen coordinates to the world coordinates of the $3d$-model. This problem is under-determined. Therefore additional constraints are introduced

to obtain a reasonable location in the $3d$-world coordinate system of the geological model. One constraint is the limitation of the manipulator to a normal vector of the function φ at the location of a control point before its displacement.

Shape transformation Once the user has picked a control point and its movement is restricted by a manipulator, the model can be deformed. During the displacement of a control point several tasks are performed.

- Computation of the new $3d$-world coordinates of the displaced control point.
- The linear system is updated and the edited function φ^* is computed.
- Due to the modified φ^*, the iso-value surface $\mathscr{S}$ is obsolete and an updated iso-value surface $\mathscr{S}^*$ has to be extracted.
- $\mathscr{S}^*$ has to be rendered in the $3d$-camera.

For each mouse event, this loop is run during the manipulation. For very large models, interactive modeling is becoming more difficult and the model update can be restricted to be performed only once after the termination of the manipulator.

Manipulator termination Once the manipulation of the model is terminated φ and $\mathscr{S}$ are re-computed one last time. Now additional control points can be defined or existing control points deleted. New manipulations can be performed on the same or on other control points.

Resource deallocation When the model edit process terminates, allocated resources for the linear system are released.

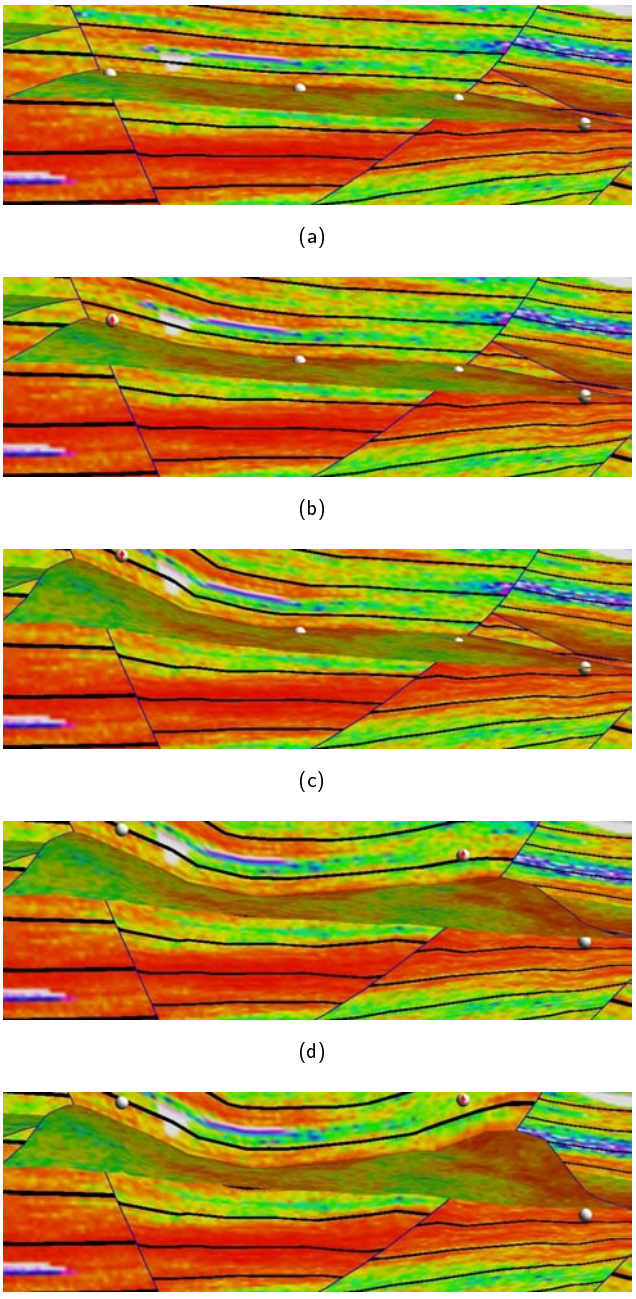
Figure 6.4 shows a series of figures where a horizon is deformed by displacing control points using the introduced workflow.

6.3.1 Real-Time Matrix-DSI (RtMxDSI)

To compute the implicit function on a tetrahedral mesh the matrix formulation of the Discrete Smooth Interpolation (DSI) – introduced in chapter 1.1.2 – is used. This section extends the concept of matrix DSI (MxDSI) with real-time capabilities. As mentioned in section 6.3 the initial interpolation of φ is based on reference data like horizons and additional control points define the deformation of the iso-value surface $\mathscr{S}$ of the implicit function φ. The reference input data as well as the control points used for shape editing are ordinary control point constraints. In general, only a very few of these additional control points are used and the number of control points of the input data is several orders of magnitude bigger. The basic idea behind real-time matrix DSI (RtMxDSI) is the decomposition of the set of constraints $\mathscr{C}$ into two subsets. The base constraints $\mathscr{C}_B$ describe the constraints of the static input data and the float constraints $\mathscr{C}_F$ are the dynamic constraints used for deformation with $\mathscr{C} = \mathscr{C}_B \cup \mathscr{C}_F$, so we can write for the decomposition of matrix $\mathbf{A}$ and vector $\mathbf{b}$:

$$\mathbf{A} = \mathbf{A}_B + \mathbf{A}_F \tag{6.1}$$

$$\mathbf{b} = \mathbf{b}_B + \mathbf{b}_F \tag{6.2}$$



(a)

(b)

(c)

(d)

(e)

Figure 6.4: This series of figures shows a cross-section and a horizon extracted from a tetrahedral mesh. The surfaces are co-rendered with a stochastic simulation [Alapetite et al., 2005; Leflon, 2005]. The fault block boundaries are highlighted in blue and the black outlines represent iso-contours of a geological time parametrization based on GeoChron. The white spheres – arbitrarily located in the $3d$-Euclidean space – are float constraints to control the shape of the horizon. Figure (a) shows the initial state of the model. In figure (b) and (c) the left control point was displaced along a straight line orthogonal to the interpolated horizon. Hereby, the horizon was deformed. Figure (d) and (e) show the same for the right control point. The geological time parametrization was updated only in a region of influence. The bottom of the model (location of the reference horizons) and the adjacent fault block were not altered by this modeling. One can also see that the deformed iso-T contours and the updated geological property remains consistent throughout the deformation. Data by courtesy of Total.

Together with equation 1.21 we can develop further:

$$\left| \begin{array}{rcl} \mathbf{A}_B & = & \sum_{c \in \mathscr{C}_B^s} A_c \cdot A_c^t \\ \mathbf{A}_F & = & \sum_{c \in \mathscr{C}_F^s} A_c \cdot A_c^t \end{array} \right. \quad \text{and} \quad \left| \begin{array}{rcl} \mathbf{b}_B & = & \sum_{c \in \mathscr{C}_B^s} b_c \cdot A_c \\ \mathbf{b}_F & = & \sum_{c \in \mathscr{C}_F^s} b_c \cdot A_c \end{array} \right. \tag{6.3}$$

So we can re-write the minimization problem from equation 1.22 as follows:

$$LS(\varphi) = \frac{1}{2} \cdot \varphi^t \cdot (\mathbf{A}_B + \mathbf{A}_F) \cdot \varphi - (\mathbf{b}_B + \mathbf{b}_F)^t \cdot \varphi \tag{6.4}$$

The implementation of this extension is realized as follows. The reference input data define the matrix $\mathbf{A}_B$ and the vector $\mathbf{b}_B$ of the base system of linear equations. The solution of this system of linear equations is the initial solution φ_0. This computation is performed as initialization. The float constraints used for deformation define the matrix $\mathbf{A}_F$ and the vector $\mathbf{b}_F$ of the float system. When the state of the float system is changed, for example by adding, removing or displacing a float constraint, the base system and the float system are merged to the common system of linear equations (see equation 6.4) and the solution of this system results in a modified solution φ^*. This decomposition enables interactive frame rates due to several aspects:

- If the state of a float constraint has changed only the float system has to be updated.
- Merging both systems is extremely fast.
- The re-interpolation of φ converges very fast – using the Conjugate Gradient method – because the new solution φ^* is very close to the initial solution of φ.

The matrix $\mathbf{A}_F$ and the vector $\mathbf{b}_F$ of the float system are extremely sparse and of course only the non zero coefficients affected by the float constraints are stored.

6.3.2 *2d*-Constrained Manipulation

The introduced method to deform implicitly defined shapes is used to edit and model geological horizons in real-time. The deformation of a horizon is based on the displacement of control points. Two kinds of control points can be distinguished. Control points can be used as tacks to fix a horizon to certain locations. These control points do not change their position. Other points can be displaced and attract like a magnet the implicitly defined horizon (see figure 6.7). The manipulator that determines the deformation is triggered by mouse pointer movements. This implies that $2d$-screen coordinates have to be traced back to $3d$-world coordinates of the model. That is an under-determined problem and we have to supply additional information to receive a reasonable deformation of the model. The under-determination is resolved by constraining the movement of the manipulator.

The developed manipulator is normal constrained. Applying this manipulator, the initial control points are coplanar with the implicitly defined horizon to be edited. The point that is moved causes a deformation of the horizon; all other control points act as tacks but can be displaced by another pick. The movement of a picked control point is hereby restricted to a normal of the modified property at the location of the control point (see figure 6.5). The beam of the normal on which the control point can be moved is further constrained by faults and reference horizons.

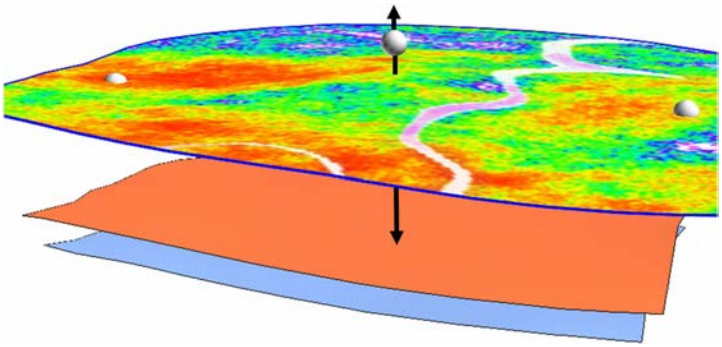


Figure 6.5: This figure shows three horizons. The horizons at the bottom (blue and red) are reference input data and the horizon – co-rendered with a stochastic simulation – is an iso-value surface extracted from a tetrahedral mesh. On that implicit surface three control points are defined and the control point in the center of the surface is moved along a normal vector of the implicit function at that location. Data by courtesy of Total.

6.3.3 Subgraph Methods

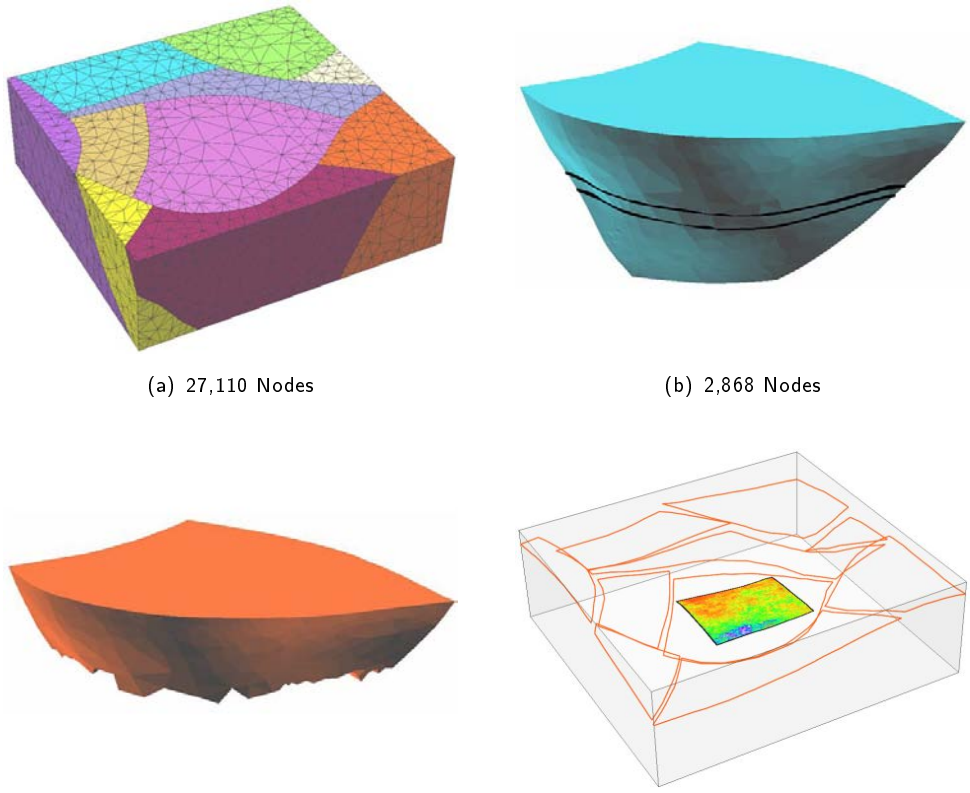
The standard DSI implementation operates on all nodes of the tetrahedral mesh. Most model editing operations only affect regional delimited subsets of nodes of this graph. Additional performance can be gained by restricting the re-interpolation of the implicit function $\varphi(x,y,z)$ to an induced subgraph $\mathscr{G}^*$ containing the set of nodes Ω^*. Ω^* contains all nodes of the tetrahedral mesh where values of $\varphi(x,y,z)$ are changed by model editing plus all nodes that are needed to define a minimum set of constraints $\mathscr{C}^* \subset \mathscr{C}$. $\mathscr{C}^*$ contains all constraints of $\mathscr{C}$ to define a function $\varphi^*(x,y,z)$ on Ω^* that is identical to $\varphi(x,y,z)$ projected on $\mathscr{G}^*$. This defines a locally restricted discrete model $\mathscr{M}^*(\Omega^*,N,\varphi^*,\mathscr{C}^*)$. In the following, different strategies to define subgraphs for local property editing are elaborated.

Topological Decomposition

Many geological models are decomposed into independent fault blocks (see figure 6.6(a)). A deformation of a horizon should only modify the horizon inside the fault block were the manipulation is carried out. The parts of this horizon in the other blocks must not be changed at the same time. Thus, an obvious subgraph is a graph only defined for the sub-volume (fault block) that is edited (see figure 6.6(b)).

Co-domain Decomposition of φ

The reference input data that are used to define the initial implicit function φ have to be honored by the property editing process. In the following only continuous functions are considered and thus a control point used for manipulating φ may never cross this reference data. As a consequence all regions beyond the input data can be considered as constant and can be excluded from the interpolation. Figure 6.7 shows three cases with two reference horizons. The control point used to deform an interpolated horizon is moved along the normal vector of φ at the initial location of the control point. In figure 6.7(a) the



(a) 27,110 Nodes (b) 2,868 Nodes

(c) 1,744 Nodes (d) 752 Nodes

Figure 6.6: Subgraph methods: (a) shows the different fault blocks of a tetrahedra tessellated model. Each fault block can be considered as independent graph. (b) shows a single fault block of the aggregate model (a) where the model editing can be reduced by one order of magnitude. Traces of the reference horizons are highlighted with black outlines. Further improvement can be achieved by restricting the editing to the region beyond the reference input data (c). (d) shows a horizon of the same model were only a certain interval of the paleo-geographic coordinates u and v are rendered. The horizon deformation is only performed on this surface patch. Data by courtesy of Total.

region of interest lies above, in figure 6.7(b) between and in figure 6.7(c) beneath the two reference horizons. This is a decomposition of the co-domain of φ excluding the regions fixed by the reference input data (see figure 6.6(c)).

Domain Decomposition of φ

The domain of φ can be decomposed in several ways. A first approach is to divide the x, y and z coordinates of the Euclidean space into intervals. For most applications like geological modeling this is generally not reasonable. A more appropriate geological decomposition is the breakdown of the model into sub-volumes using the paleo-geographic coordinates u, v obtained by the GeoChron parametrization. Figure 6.6(d) shows a horizon where only the geology inside a certain interval of u and v is rendered. The regions outside this interval are transparent and are excluded from the horizon deformation. A more general decomposition is the restriction of the volume

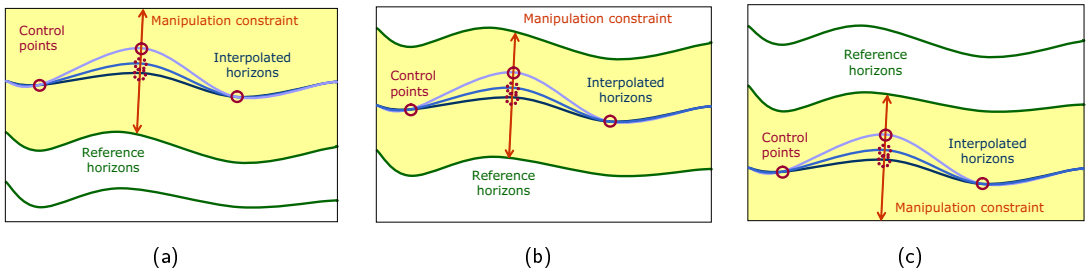


Figure 6.7: This series of figures shows two reference horizons (green) used as input data, an interpolated horizon (blue) and three control points (red) to edit the interpolated horizon. The control point in the center is moved along a normal vector of the interpolated horizon at the initial position of the control point. The arrangement of the reference data and the interpolated horizon define a zone of interest (yellow). For case (a) the updated region lies above, for case (b) between and for case (c) beneath the two reference horizons.

of interest by distance transforms. Hereby, only a volume within a certain distance to objects like faults, wells, control points or manipulators is affected by property editing.

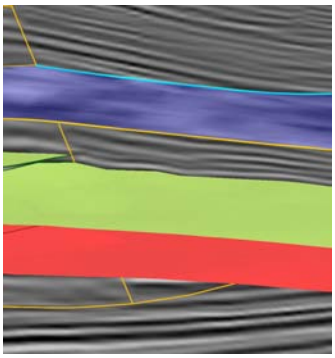
In general the subgraph $\mathscr{G}^*$ contains much less nodes than the graph of the complete model $\mathscr{G}$. Thus, the performance for the re-interpolation needed to edit φ is increased tremendously. Of course, the introduced methods to build $\mathscr{G}^*$ can be combined.

6.3.4 Example of Application

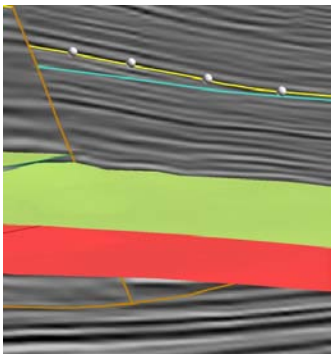
Figure 6.8 shows a cross-section and a horizon co-visualized with a seismic cube. The reference horizons are painted red and green and the fault block boundaries are outlined in orange. One can see in figure 6.8(a) that the intersection of the interpolated horizon with the cross-section (turquoise curve) does not exactly follow the seismic reflector. To fit the interpolated horizon to the seismic reflector we use normal constrained deformation and create several control points on the initial interpolated horizon. The fitting of some of these control points to the seismic reflector deforms the interpolated horizon. The updated intersection between the re-interpolated horizon and the cross-section (yellow curve) matches the seismic reflector exactly (see figure 6.8(b) and 6.8(c)). The model editing is performed in real-time with immediate user feedback at interactive frame rates.

6.4 Summary

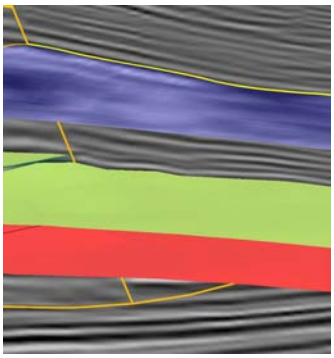
This chapter presented a new, interactive method to deform surfaces like geological horizons in real-time. This method is based on a tri-variate implicit function $\varphi(x,y,z)$ that was interpolated from reference input data. A surface can be considered as an iso-value contour where $\varphi(x,y,z)$ takes the constant value ϕ so $\varphi - \phi = 0$. If the implicit function $\varphi(x,y,z)$ is modified the iso-value surface $\mathscr{S}$ is automatically deformed. The manipulation of the model is performed by displacing control points that carry the same



(a) The intersection of the interpolated horizon with the cross-section (turquoise curve) shows that the horizon does not match the reflector precisely.



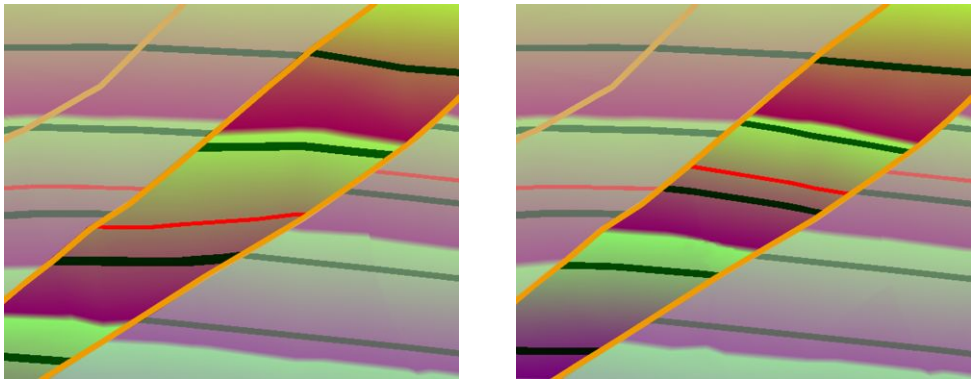
(b) By displacing control points the interpolated horizon (not illustrated) is deformed and the updated intersection (yellow curve) of this horizon with the cross-section matches the seismic reflector exactly.



(c) The re-interpolated horizon goes exactly through the seismic reflector. Surface editing is performed interactively with immediate user feedback at interactive frame rates.

Figure 6.8: This series of figures illustrates a cross-section and an interpolated horizon (blue) extracted from a tetrahedral mesh co-rendered with a seismic cube. The red and green horizons are the reference horizons of this model. The fault block boundaries are outlined in orange. The control points are moved along a straight line orthogonal to the interpolated horizon. Data by courtesy of Total.

property value as $\mathscr{S}$. Displaced control points edit $\varphi(x,y,z)$ and not directly $\mathscr{S}$. The deformed surface $\mathscr{S}^*$ is just a re-computed iso-value surface and is not defined explicitly. Hereby artifacts like bumps or unnatural curvatures known from Free-Form Deformation applied on explicit triangulated surfaces do not occur. Topological constraints like the non-intersection of horizons and contacts between faults and horizons are fulfilled automatically. Control points that are not displaced hold the values of the implicit function at their location and can be considered as tacks. The attraction of a control point can be defined by specifying a relative weight. Discontinuities in the tetrahedral mesh are dissociated nodes in the corresponding graph. Thus a modification of a function only affects one side defined by the unconformity. Figure 6.9(a) shows a cross-section through a model with several normal faults. The GeoChron time parametrization of the center region was edited in figure 6.9(b). The time function beyond the two normal faults bounding this region remained unchanged. In a typical application model modifications have local character. By the definition of an induced subgraph $\mathscr{G}^* \subset \mathscr{G}$ the number of nodes on which the function editing is performed is decreased tremendously. By this improvement the deformations can be performed with direct user feedback at interactive frame rates.



(a) (b)

Figure 6.9: (a) shows a cross-section through a GeoChron model with several normal faults outlined in orange. The property is a GeoChron time parametrization painted with a periodic colormap and iso-contours. The intersection with an interpolated horizon is outlined in red. (b) shows the same model where the time function was edited in the central region. The time function in the other regions beyond the bounding normal faults of the modified region was not changed.

Conclusions and Perspectives

As shown in the introduction, tetrahedral meshes are used more and more frequently for geo-modeling applications. For example restoration, flow simulation, characterization of fractured reservoirs and the GeoChron model make massive use of simplicial complexes and the developed tools. This work developed new algorithms for geological visualization and modeling of tetrahedral meshes. The contributions of this thesis were decomposed into the following.

Visualization of Tetrahedral Meshes

The visualization of tetrahedral meshes has been limited to the boundary representation – inside the G◯CAD project – so, the first part of this work developed new methods for visualizing this type of mesh with special focus on retrieval of geological information. By these algorithms, the extraction and combination of complex geological information of different nature is now possible on tetrahedral meshes.

Iso-value Surface Interpolation

An iso-value surface is built by extracting a level set where a property takes a constant value. The polygonization of this subset is performed on the CPU (Central Processing Unit). The intersection computation between an iso-value surface and the tetrahedral mesh is accelerated by an octree in a parametric value-space. Further improvement of the performance is gained on symmetric multiprocessing or multi-core systems, respectively, by a parallelization of the iso-value surface extraction algorithms. Parallelization is a core technique of modern and future CPUs. Only parallelized software – as implemented for this work – can profit from this development. The model (geometric information) and the view (rendering information) are separated into different display lists, which are dynamic high-performance interfaces to the graphics hardware. The implementation is based on generic design patterns to ease further extensions. The first extensions to visualize non-scalar properties were already implemented by Massenet [2005]. The developed methods reach interactive frame rates even on models with several hundred thousands tetrahedrons on standard consumer hardware. Thus, the model size – used in geo-modeling – is no longer limited by the visualization but by the numerical complexity of the particular application. Common models can be explored at interactive frame rates.

Advanced Rendering for Geo-Modeling

Beside the geometric information of iso-value surfaces, the rendering of heterogenous geological information was investigated. Hereby, different information – stored in texture maps or retrieved from topology – is blended and combined on the GPU (Graphics Processing Unit). The blending is performed by built-in blending functions or user-defined fragment shaders. In both cases the image is created in one rendering path without additional computations on the CPU. Information like stratigraphic columns, distance maps, iso-contours and geological/geophysical data are combined. Many methods like geo-statistics, image processing (distance transforms, convolution, ...) and many more perform better on a structured domain like a Cartesian grid. The results of this computations – performed on a structured domain – are mapped to the unstructured tetrahedral mesh by texture mapping. Further, multitexturing is used for boolean CSG (Constructive Solid Geometry) operations – performed on implicitly defined models. The developed method has constant complexity $\mathcal{O}(1)$ independent from size and complexity of the models. These high-performance boolean operators can be used for complex conditional $3d$-GIS queries. The outline of faults and fault block boundaries are computed on the fly from the micro-topology of the tetrahedral mesh.

Volume rendering and hardware accelerated visualization of non-scalar properties are subjects for future investigation. User-defined blending functions implemented in fragment shaders can be extended to select and classify information on the GPU. The visualization methods are the basis for the developed modeling algorithms.

Modeling of Tetrahedral Meshes

Modeling methods are subdivided into two groups. *Geometric* modeling optimizes the numerical resolution of tetrahedral meshes by local mesh refinement. *Implicit* modeling addresses the creation and manipulation of implicit models defined on tetrahedral meshes.

Geometric Modeling

Three methods for local refinement of tetrahedral meshes were implemented and evaluated for applications using the DSI (Discrete Smooth Interpolation) method. These methods are: point insertion honoring the Delaunay criterion, edge bisection and template-based cell decomposition. As the DSI method is very stable even on degenerated cells, shape factors of the generated simplices are not optimized. Generally, the number of simplices is a critical factor for numerical methods and should be minimized. Thus, point insertion based on the Delaunay criterion is the first choice. If the numerical methods tolerate large meshes, template-based decomposition is an alternative with a fast convergence. Hybrid refinement was also investigated. The first refinement levels are performed using Delaunay refinement. Afterwards, template-based decomposition is only applied in bounded regions. The hybrid method demonstrates a way to generate large cell size gradients with good convergence. These refinement methods obviate model rebuilds after

an update of the input data and tremendously speed up the model generation and update process.

The developed methods were applied to applications that use the DSI method. As mentioned before, DSI is very robust in regard to mesh quality. If the refinement methods should be applied for finite element methods or other methods that depend on the mesh quality, shape factors for the generated cells have to be optimized during refinement. Possible methods are edge switches, mesh smoothing or the already discussed point insertion. Local mesh refinement was applied to increase the numerical resolution of tetrahedral meshes used for surface reconstruction.

Implicit Modeling

Implicit modeling covers the definition as well as the manipulation of implicit models. Geological interfaces like horizons or salt-top surfaces are extracted from seismic data. The presented work devised a new method to reconstruct complex surfaces from point clouds obtained form seismic surveys. Common surface reconstruction methods – optimized for $3d$-scan data – cannot be applied because of the properties of geological data. Geological interfaces like horizons or salt-top surfaces may have discontinuities and boundaries. Further, the point data obtained by seismic surveys are extremely noisy, can have sparse regions and have no uniform distribution. The developed method takes all these difficulties into account. To do so, an implicit function is interpolated on a tetrahedral mesh. The input data carry the value zero. The two sides of the surface to be reconstructed are user-defined by assigning a different mathematical sign to these regions. The iso-value *zero* of the interpolated function defines the reconstructed surface. Unconformities are introduced by discontinuities in the tetrahedral mesh. The stiffness of the surface efficiently controls its tolerance to noise and is also a user-defined parameter. The developed method does not have to compute normal vectors at any stage of its algorithm and obtains the orientation by interactive user input. Thus, it can be applied to noisy and sparse data sets, contrary to common methods that evaluate normal vectors. In the same way, ambiguous shapes can be resolved interactively by the user. This robust and efficient surface reconstruction method was applied to real-world data sets with up to 400,000 data points.

The definition of the regions and discontinuities are user-defined. Especially the region definitions require a certain experience of the user. An automated process for the region definition and a guided workflow that integrates the local mesh refinement methods would increase the usability of the developed reconstruction method in an industrial environment.

The second implicit modeling technique addresses the manipulation of implicitly defined models especially the rapid manipulation of horizons (GeoChron time parametrization). New insights or the knowledge of an engineer might require model updates. Numerical information can be incorporated by time-expensive model rebuilds. But personal experience is very difficult to describe in numerical algorithms. The presented work developed a new, intuitive way for the interactive manipulation of horizons integrating all previous results of this thesis. Therefore a set of control points for the GeoChron time

parametrization is created. Displaced control points manipulate the implicit horizons (iso-value surfaces of the GeoChron time parametrization). This method offers a real $3d$-manipulation of the model and guarantees geological constraints like horizon-fault contacts and prevents neighboring horizons from intersecting each other. So, complex geological constraints are fulfilled automatically. The manipulation is performed with immediate user feedback and reaches interactive frame rates even for large models by restricting the manipulation to regions of interest. All associated geological properties are updated automatically in a geologically consistent way. The immediate user-feedback in three dimensions gives the user full control over his manipulations.

Further work can be done in the field of preserving local features during manipulation. Also the extension to other interfaces like salt-top surfaces computed by the introduced reconstruction method is interesting. At the moment the manipulation is constrained to normal vectors of the initial horizon. Many other types of manipulators are imaginable.

Summary

The presented work introduced for the first time a set of advanced visualization and modeling tools for tetrahedral meshes into the GOCAD project. One of the main focuses were generic design and extensibility. The developed methods are a set of intuitive and interactive tools for everyday work with tetrahedral meshes in geo-modeling. The developed tools open a new door to understand complex geological information from different sources in one model. New, interactive modeling techniques give the user full control over model creation and manipulation. These tools can easily be adapted and extended for many applications such as the extension of the visualization to non-scalar properties for restoration. The results of this work are already used in several recent research projects and will soon be available for industrial applications.

Bibliography

ADAMSON, A. AND ALEXA, M. 2003. Approximating and intersecting surfaces from points. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, 230–239.

ALAPETITE, J., LEFLON, B., GRINGARTEN, E., AND MALLET, J.-L. 2005. Stochastic modeling of fluvial reservoirs: the YACS approach. In *SPE 97271*. Dallas, Texas, USA, 5p.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01*. IEEE Computer Society Press, Washington, DC, USA, 21–28.

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Transactions on Graphics 24,* 3, 617–625.

AMENTA, N., CHOI, S., AND KOLLURI, R. K. 2001. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications 19,* 2/3, 127–153.

ANGEL, E. 2003. *Interactive Computer Graphics, A Top-Down Approach*, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

APEL, M. 2004. A 3d geoscience information system framework. Ph.D. thesis, INPL Nancy and TU Freiberg.

ARNOLD, D. N., MUKHERJEE, A., AND POULY, L. 2000. Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing 22,* 2, 431–448.

BAJAJ, C. 1999. *Data Visualization Techniques*. John Wiley & Sons Ltd, Indianapolis, IN, USA, Chapter Accelerated IsoContouring of Scalar Fields.

BAJAJ, C. L., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3d scans. In *ACM SIGGRAPH 1995, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 109–118.

BARTZ, D. AND STRASSER, W. 1999. Asynchronous parallel construction of recursive tree hierarchies. In *ParNum '99: Proceedings of the 4th International ACPC Conference*. Springer-Verlag, London, UK, 427–436.

BENTLEY, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM 18,* 9, 509–517.

BERNARDINI, F., BAJAJ, C. L., CHEN, J., AND SCHIKORE, D. 1999. Automatic reconstruction of 3d CAD models from digital scans. *International Journal of Computational Geometry and Applications 9,* 4/5, 327–369.

BERZINS, M. 1999. Mesh quality: A function of geometry, error estimates or both? *Engineering with Computers 15,* 3, 236–247.

BLOOMENTHAL, J. AND WYVILL, B. 1990. Interactive techniques for implicit modeling. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*. ACM Press, New York, NY, USA, 109–116.

BLOOMENTHAL, J. AND WYVILL, B. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Fransisco.

BORREL, P. AND RAPPOPORT, A. 1994. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics 13,* 2, 137–155.

BOTSCH, M. AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. In *Eurographics 2005, Computer Graphics Forum*. 611–621.

BOUBEKEUR, T. AND SCHLICK, C. 2005. Generic mesh refinement on GPU. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM Press, New York, NY, USA, 99–104.

BOWYER, A. 1981. Computing dirichlet tessellations. *Computer Journal 24,* 2, 162–166.

BUATOIS, L., CAUMON, G., AND LÉVY, B. 2006. GPU accelerated isosurface extraction on complex polyhedral grids. In *26th Gocad Meeting Proceedings*. Nancy, France.

CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *ACM SIGGRAPH 2001, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 67–76.

CARR, J. C., BEATSON, R. K., MCCALLUM, B. C., FRIGHT, W. R., MCLENNAN, T. J., AND MITCHELL, T. J. 2003. Smooth surface reconstruction from noisy range data. In *GRAPHITE '03*. ACM Press, New York, NY, USA, 119–ff.

CASTANIÉ, L., LÉVY, B., AND BOSQUET, F. 2005. VolumeExplorer: Roaming large volumes to couple visualization and data processing for oil and gas exploration. In *Procceedings of IEEE Visualization '05*. IEEE Computer Society Press, Washington, DC, USA.

CAUMON, G. 2003. Représentation, visualisation et modification de modèles volumiques pour les géosciences. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

CAUMON, G., LEVY, B., CASTANIE, L., AND PAUL, J. 2005. Advanced visualization for various unstructured grids using ad hoc topological structures. *Computers and Geosciences 31,* 6 (September), 671–680.

CAUMON, G., SWORD, C. H., AND MALLET, J.-L. 2003. Constrained modifications of non-manifold b-rep models. In *Proc. 8th ACM Symposium on Solid Modeling and Applications*. ACM Press, New York, NY, 310–315.

CAUMON, G., SWORD, C. H., AND MALLET, J.-L. 2004. Interactive editing of sealed geological 3d models. *Mathematical Geology 36,* 4, 405–424.

CEBRAL, J., CASTRO, M., SOTO, O., LÖHNER, R., AND ALPERIN, N. 2003. Blood-flow models of the circle of willis from magnetic resonance data. *Journal of Engineering Mathematics 47,* 3/4, 369–386.

CHEN, M., SILVER, D., WINTER, A. S., SINGH, V., AND CORNEA, N. 2003. Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In *VG '03: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*. ACM Press, New York, NY, USA, 35–44.

CHEW, L. P. 1987. Constrained delaunay triangulations. In *SCG '87: Proceedings of the third annual symposium on Computational geometry*. ACM Press, New York, NY, USA, 215–222.

CHUA, C. AND NEUMANN, U. 2000. Hardware-accelerated free-form deformation. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. ACM Press, New York, NY, USA, 33–39.

CIGNONI, P., MARINO, P., MONTANI, C., PUPPO, E., AND SCOPIGNO, R. 1997. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics 3,* 2, 158–170.

CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. 1998. Tetrahedra based volume visualization. In *Mathematical Visualization*, H.-C. Hege and K. Polthier, Eds. Springer-Verlag, Heidelberg, 3–18.

CONRAUD, J. 1997. Génération de maillages de simplexes pour la modélisation d'objets naturels. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

CONREAUX, S. 2001. Modélisation de 3-variétés à bases topologique: application à la géologie. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

COQUILLART, S. 1990. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*. ACM Press, New York, NY, USA.

CUISENAIRE, O. 1999. Distance transformations: Fast algorithms and applications to medical image processing. Ph.D. thesis, Universite Catholique de Louvain, Louvain-la-Neuve, Belgium.

CURLESS, B. AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *ACM SIGGRAPH 1996, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 303–312.

CUTLER, B., DORSEY, J., AND MCMILLAN, L. 2004. Simplification and improvement of tetrahedral models for simulation. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 93–102.

DEY, T. K. AND SUN, J. 2005. An adaptive MLS surface for reconstruction with guarantees. In *Eurographics Symposium on Geometry Processing*. ACM Press, New York, NY, USA, 43–52.

EDELSBRUNNER, E. 1980. Dynamic data structures for orthogonal intersection queries. Tech. rep., Institut für Informationsverarbeitung, Technische Universität, Graz, Austria.

EDELSBRUNNER, H. AND MÜCKE, E. P. 1994. Three-dimensional alpha shapes. *ACM Transactions on Graphics 13,* 1, 43–72.

FERNANDO, R. AND KILGARD, M. J. 2003. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. T. 2005. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics 24,* 3, 544–552.

FRANK, T. AND MALLET, J.-L. 2004. Exploring the tetrahedral decomposition of the subsurface. In *24th Gocad Meeting Proceedings*. Nancy, France.

FREITAG, L. A. AND OLLIVIER-GOOCH, C. 1996. A comparison of tetrahedral mesh improvement techniques. In *Fifth International Meshing Roundtable*. Sandia National Laboratories, Pittsburgh, Pennsylvania, 87–106.

GALLAGHER, R. 1991. Span filtering: an optimization scheme for volume visualization of large finite element models. In *Proceedings of the 2nd conference on Visualization '91*. IEEE Computer Society Press, Washington, DC, USA, 68–75.

GEORGE, P. AND BOROUCHAKRI, H. 1997. *Triangulation de Delaunay et maillage*. Hermes, Paris.

GÉRAUD, T. AND DURET-LUTZ, A. 2000. Generic programming redesign of patterns. In *Proceedings of the 5th European Conference on Pattern Languages of Programs (EuroPLoP'2000)*. Irsee, Germany.

GILES, M. AND HAIMES, R. 1990. Advanced interactive visualization for CFD. *Computing Systems in Engineering 1,* 1, 51–62.

GILL, P. E., MURRAY, W., AND WRIGHT, M. H. 1982. *Practical Optimization*. Academic Press, London and New York.

GRIMM, C. M. AND HUGHES, J. F. 1995. Modeling surfaces of arbitrary topology using manifolds. In *ACM SIGGRAPH 1995, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 359–368.

GROSSE, O. 2002. Remise en cohérence d'un modèle géologique 3D. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

HESTENES, M. R. AND STIEFEL, E. 1952. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand. 49*, 409–436.

HINTON, E. AND OWEN, D. R. J. 1979. *An Introduction to Finite Element Computations*. Pineridge Press, Swansea. 385 pp.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*. ACM Press, New York, NY, USA, 71–78.

KARPENKO, O., HUGHES, J. F., AND RASKAR, R. 2002. Free-form sketching with variational implicit surfaces. *Computer Graphic Forum 21,* 3.

KILGARD, M. J. 2004. *NVIDIA OpenGL Extension Specifications*. NVIDIA Corporation.

KLEIN, J. AND ZACHMANN, G. 2004. Proximity graphs for defining surfaces over point clouds. In *Symposium on Point-Based Graphics*. ETHZ, Zürich, Switzerland.

KLEIN, T., STEGMAIER, S., AND ERTL, T. 2004. Hardware-accelerated Reconstruction of Polygonal Isosurface Representations on Unstructured Grids. In *Proceedings of Pacific Graphics '04*. 186–195.

KOLLURI, R., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2004. Spectral surface reconstruction from noisy point clouds. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 11–21.

L. ENDRES, P. K. 2003. Octasection-based refinement of finite element approximations of tetrahedral meshes that guarantees shape quality. *International Journal for Numerical Methods in Engineering 59,* 1, 69–82.

LEDEZ, D. 2002. Level sets: a general framework for geological applications. In *22nd Gocad Meeting Proceedings*. Nancy, France.

LEDEZ, D. 2003. Modelisation d'objets naturels par formulation implicite. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

LEFLON, B. 2005. Modélisation des hétérogénéités lithologiques à l'échelle du réservoir pétrolier en millieu marin et fluviatile. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

LEPAGE, F. 2003. Génération de maillages tridimensionnels pour la simulation des phénomènes physiques en géosciences. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

LEVIN, D. 1998. The approximation power of moving least-squares. *Mathematics of Computation 67,* 224, 1517–1531.

LEVIN, D. 2003. *Geometric Modeling for Scientific Visualization*. Springer Verlag, Heidelberg, Germany, Chapter Mesh-independent surface interpolation, 37–49.

LEVOY, M. AND WHITTED, T. 1985. The use of points as a display primitive. Tech. Rep. 85022, Computer Science Department, University of North Carolina at Chapel Hill.

LÉVY, B., CAUMON, G., CONREAUX, S., AND CAVIN, X. 2001. Circular incident edge lists: a data structure for rendering complex unstructured grids. In *IEEE Visualization*, T. Ertl, K. Joy, and A. Varshney, Eds.

LIU, A. AND JOE, B. 1996. Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision. *Mathematics of Computation 65,* 215, 1183–1200.

LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. 1996. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics 2,* 1, 73–84.

LORENSEN, W. AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*. ACM Press, New York, NY, USA, 163–169.

MACÉ, L., MURON, P., AND MALLET, J.-L. 2005. Integration of fracture data into 3d geomechanical modeling to enhance fractured reservoirs characterization. In *SPE 95827*. Dallas, Texas, USA, 9p.

MALLET, J.-L. 1992. Discrete smooth interpolation in geometric modeling. *Computer-Aided Design 24,* 4, 178–191.

MALLET, J.-L. 2002. *Geomodeling*. Oxford University Press, New York.

MALLET, J.-L. 2003. Constraining a piecewise linear function defined on a 3d complex (applications to 3d restorarion). In *23rd Gocad Meeting Proceedings*. Nancy, France.

MALLET, J.-L. 2004. Space-time mathematical framework for sedimentary geology. *Mathematical Geology 36,* 1.

MALLET, J.-L., MOYEN, R., FRANK, T., LEFLON, B., AND ROYER, J.-J. 2004. Getting rid of stratigraphic grids. In *66th EAGE annual meeting*. Paris.

MARK, W. R., GLANVILLE, R. S., AKELEY, K., AND KILGARD, M. J. 2003. Cg: a system for programming graphics hardware in a C-like language. *ACM Transactions on Graphics 22,* 3, 896–907.

MASSENET, J. 2005. Tensorial visualization in SolidExplorer. LIAD report, unpublished.

MATTSON, T. G. 2000. An introduction to OpenMP 2.0. In *ISHPC '00: Proceedings of the Third International Symposium on High Performance Computing*. Springer-Verlag, London, UK, 384–390.

MCCREIGHT, E. 1980. Efficient algorithms for enumerating intersecting intervals and rectangles. Tech. rep., Xerox Palo Alto Research Center, Palo Alto, CA, USA.

MCGAUGHEY, J., SPRAGUE, K., DE KEMP, E., AND WONG, W. 2004. Spatial targeting using queries in a gocad-based 3-D GIS environment. In *24th Gocad Meeting Proceedings*. Nancy, France.

MEDEROS, B., AMENTA, N., VELHO, L., AND DE FIGUEIREDO, L. H. 2005. Surface reconstruction for noisy point clouds. In *SGP '05: Proceedings of the 2005 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 53–62.

MEIJSTER, A. 2004. Efficient sequential and parallel algorithms for morphological image processing. Ph.D. thesis, University of Groningen, Netherlands.

MEIJSTER, A., ROERDINK, J., AND HESSELINK, W. H. 2000. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing*. Kluwer, 331–340.

MEYERS, S. 1997. *Effective C++: 50 Specific Ways to Improve Your Programs and Design*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

MILLER, G. L., TALMOR, D., TENG, S. H., AND WALKINGTON, N. 1995. A delaunay based numerical method for three dimensions: Generation, formulation, and partition. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*.

MOORE, G. E. 1965. Cramming more components onto integrated circuits. *Electronics 38,* 8 (19 April), 114–117.

MORSE, B. S., YOO, T. S., CHEN, D. T., RHEINGANS, P., AND SUBRAMANIAN, K. R. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*. IEEE Computer Society Press, Washington, DC, USA, 89–ff.

MOYEN, R. 2005. Paramétrisation 3D en géologie sédimentaire : le modèle GeoChron. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

MURON, P. 2005. Méthodes numériques 3-D de restauration des structures géologiques faillées. Ph.D. thesis, Institut National Polytechnique de Lorraine, Nancy, France.

MURON, P., TERTOIS, A.-L., MALLET, J.-L., AND HOVADIK, J. 2005. An efficient and extensible interpolation framework based on the matrix formulation of the discrete smooth interpolation. In *25th Gocad Meeting Proceedings*. Nancy, France.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics 24,* 3.

OHTAKE, Y., BELYAEV, A., AND ALEXA, M. 2005. Sparse low-degree implicits with applications to high quality rendering. In *SGP '05: Proceedings of the 2005 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 149–158.

OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition of unity implicits. *ACM Transactions on Graphics 22,* 3, 463–470.

PASCUCCI, V. 2004. Isosurface computation made simple. In *VisSym*, O. Deussen, C. D. Hansen, D. A. Keim, and D. Saupe, Eds. Eurographics Association, 293–300.

PAULY, M., KEISER, R., KOBBELT, L. P., AND GROSS, M. 2003. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics 22,* 3, 641–650.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: surface elements as rendering primitives. In *ACM SIGGRAPH 2000*. ACM Press, New York, NY, USA, 335–342.

PLATE, J., TIRTASANA, M., CARMONA, R., AND FRÖHLICH, B. 2002. Octreemizer: a hierarchical approach for interactive roaming through very large volumes. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*. Eurographics Association, 53–ff.

PLAZA, A. AND CAREY, G. F. 1996. About local refinement of tetrahedral grids based on bisection. In *Fifth International Meshing Roundtable*. Sandia National Laboratories, Pittsburgh, Pennsylvania, 123–136.

PLAZA, A. AND RIVARA, M. C. 2003. Mesh refinement based on the 8-tetrahedra longest-edge partition. In *Twelfth International Meshing Roundtable*. Sandia National Laboratories, Santa Fe, New Mexico, 67–78.

POLLOCK, D. 1988. Semianalytical computation of path lines for finite-difference models. *Ground Water 26,* 6 (November-December), 743–750.

PRÉVOST, M., EDWARDS, M., AND BLUNT, M. 2001. Streamline Tracking on Curvilinear Structured and Unstructured Grids. *Proceedings of the 2001 SPE Reservoir Simulation Symposium, Houston, Texas, February 11-14 SPE 66347*, 15p.

REMY, N., CAUMON, G., AND LÉVY, B. 2004. Bridging the gap between the Geostatistics Template library and gocad. Application to non-scalar values on unstructured grids. In *Proceedings of the 24th Gocad meeting, Nancy*.

RÖTTGER, S., KRAUS, M., AND ERTL, T. 2000. Hardware-Accelerated Volume and Isosurface Rendering Based On Cell-Projection. In *Proceedings of IEEE Visualization '00*. IEEE, 109–116.

ROYER, J.-J. 2005. Conditional integration of a linear function on a tetrahedron. In *25th Gocad Meeting Proceedings*. Nancy, France.

RUPPERT, J. 1995. A delaunay refinement algorithm for quality 2-dimensional mesh generation. In *SODA '93: Selected papers from the fourth annual ACM SIAM symposium on Discrete algorithms*. Academic Press, Inc., Orlando, FL, 548–585.

SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNII, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14,* 4, 181–188.

SCHEIDEGGER, C. E., FLEISHMAN, S., AND SILVA, C. T. 2005. Triangulating point set surfaces with bounded error. In *SGP '05: Proceedings of the 2005 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 63–72.

SCHEIN, S. AND ELBER, G. 2004. Discontinuous free form deformations. In *Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society Press, Washington, DC, USA, 227–236.

SEDERBERG, T. W. AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*. ACM Press, New York, NY, USA.

SEGAL, M. AND AKELEY, K. 2003. *The OpenGL Graphics System: A specification (Version 1.5)*. Silicon Graphics.

SEGAL, M. AND AKELEY, K. 2004. *The OpenGL Graphics System: A specification (Version 2.0)*. Silicon Graphics.

SHEN, H.-W. AND JOHNSON, C. R. 1995. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *VIS '95: Proceedings of the 6th conference on Visualization '95*. IEEE Computer Society, Washington, DC, USA, 143.

SHEPARD, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*. ACM Press, New York, NY, USA, 517–524.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., School of Computer Science, Carnegie Mellon University.

SHEWCHUK, J. R. 1998. Tetrahedral mesh generation by delaunay refinement. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*. ACM Press, New York, NY, USA, 86–95.

SHEWCHUK, J. R. 2002. What is a good linear element? interpolation, conditioning, and quality measures. In *Eleventh International Meshing Roundtable*. Sandia National Laboratories, Ithaca, NY, 115–126.

SHIRLEY, P. AND TUCHMAN, A. 1990. A polygonal approximation to direct scalar volume rendering. In *Proceedings of the 1990 workshop on Volume visualization*. ACM Press, New York, NY, USA, 63–70.

SIEK, J. AND LUMSDAINE, A. 1998. The matrix template library: A generic programming approach to high performance numerical linear algebra. In *ISCOPE*. 59–70.

STEPANOV, A. A. AND LEE, M. 1994. The Standard Template Library. Tech. Rep. X3J16/94-0095, WG21/N0482.

STROUSTRUP, B. 2000. *The C++ Programming Language*, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

TERTOIS, A.-L., FRANK, T., AND MALLET, J.-L. 2005. Real-time tetrahedral volume editing accounting for discontinuities. In *9th International Conference on Computer Aided Design and Computer Graphics*. IEEE Computer Society Press, Hong Kong, China.

TERTOIS, A.-L. AND MALLET, J.-L. 2005. Surface reconstruction and fitting from dense point sets. In *25th Gocad Meeting Proceedings*. Nancy, France.

TURK, G. AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *ACM SIGGRAPH 1994, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 311–318.

TURK, G. AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *ACM SIGGRAPH 1999, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 335–342.

TURK, G. AND O'BRIEN, J. F. 2002. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21,* 4, 855–873.

WATSON, D. F. 1981. Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal 24*, 167–171.

WEILER, K. 1988. The radial edge structure: a topological representation for non-manifold geometric boundary modeling. In *Geometric Modeling for CAD applications*. Amsterdam, North-Holland.

WESTERMANN, R. AND ERTL, T. 1998. Efficiently using graphics hardware in volume rendering applications. In *ACM SIGGRAPH 1998, Computer Graphics Proceedings*. ACM Press, New York, NY, USA, 169–177.

WILHELMS, J. AND VAN GELDER, A. 1990. Octrees for faster isosurface generation. In *Proceedings of the 1990 workshop on Volume visualization*. ACM Press, New York, NY, USA, 57–62.

WITKIN, A. P. AND HECKBERT, P. S. 1994. Using particles to sample and control implicit surfaces. In *ACM SIGGRAPH 1994, Computer Graphics Proceedings*. ACM Press, New York, NY, USA.

XIE, H., MCDONNELL, K. T., AND QIN, H. 2004. Surface reconstruction of noisy and defective data sets. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*. IEEE Computer Society Press, Washington, DC, USA, 259–266.

YING, L. AND ZORIN, D. 2004. A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Transactions on Graphics 23,* 3, 271–275.

YOSHIZAWA, S., BELYAEV, . G., AND SEIDEL, H.-P. 2002. A simple approach to interactive free-form shape deformations. In *Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society Press, Washington, DC, USA.

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Heinz KONIETZKY, Professeur, Université Bergakademie Freiberg, Allemagne

Monsieur Klaus SPITZER, Professeur, Université Bergakademie Freiberg, Allemagne

Monsieur Jean-Paul CHILÈS, Professeur, Centre de Géostatistiques, ENSMP, Fontainebleau

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur FRANK Tobias

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Advanced Visualization and Modeling of Tetrahedral Meshes"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Géosciences** »

Fait à Vandoeuvre, le 21 mars 2006
Le Président de l'I.N.P.L.,
L. SCHUFFENECKER

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55

Abstract Tetrahedral meshes are becoming more and more important for geo-modeling applications. The presented work introduces new algorithms for efficient *visualization* and *modeling* of tetrahedral meshes. *Visualization* consists of a generic framework that includes the extraction of geological information like stratigraphic columns, fault block boundaries, simultaneous co-rendering of different attributes and boolean operations of Constructive Solid Geometry with constant complexity. *Modeling* can be classified into *geometric* and *implicit* modeling. *Geometric* modeling addresses local mesh refinement to increase the numerical resolution of a given mesh. *Implicit* modeling covers the definition and manipulation of implicitly defined models. A new surface reconstruction method was developed to reconstruct complex, multi-valued surfaces from noisy and sparse data sets as they occur in geological applications. The surface can be bounded and may have discontinuities. Further, this work proposes a new and innovative algorithm for rapid editing of implicitly defined shapes like horizons based on the GeoChron parametrization. The editing is performed interactively on the $3d$-volumetric model and geological constraints are respected automatically.

Résumé L'approche volumique (par maillages tétraédriques) est de plus en plus utilisée en modélisation géologique. La motivation de ce travail est donc de développer des algorithmes performants pour la *visualisation* et la *modélisation* de maillages tétraédriques. Concernant la *visualisation*, un ensemble d'outils génériques a été développé, permettant l'extraction d'informations géologiques, telles que les colonnes stratigraphiques ou les frontières de blocs de failles. Le rendu simultané de différents attributs et des opérations booléennes de construction géométrique de solides complexes est également devenu possible. Concernant la *modélisation*, deux axes de recherche ont été explorés: la modélisation *géométrique* et la modélisation *implicite*. La modélisation *géométrique* a consisté à améliorer la résolution numérique d'un maillage donné par raffinement local, selon différentes stratégies possibles. La modélisation *implicite* a permis à la fois la construction et la manipulation de modèles définis implicitement. Une nouvelle méthode de reconstruction de surface a été développée à partir des fonctions implicites. Cette méthode a été testée avec succès pour la reconstruction de surfaces complexes telles qu'on les rencontre en géologie: grand nombre de données, données éparses et bruitées, surfaces multivaluées ou pouvant avoir des bords libres. Enfin, un algorithme d'édition rapide d'horizon a été développé. Cet algorithme, qui intègre beaucoup des développements de cette thèse, travaille sur la forme implicite des horizons, basée sur la paramétrisation Geochron. Cette édition est réalisée de façon intéractive sur le modèle $3d$, tout en honorant automatiquement les contraintes géologiques.

Zusammenfassung Tetraedergitter gewinnen immer mehr an Bedeutung für die Geomodellierung. Die vorliegende Arbeit stellt neue Algorithmen für die effiziente *Visualisierung* und *Modellierung* von Tetraedergittern vor. Die *Visualisierung* besteht aus einer generischen Architektur zur geologischen Informationsgewinnung mittels Visualisierung stratigraphischer Abfolgen, Störungsflächen, Co-Visualisierung diverser Attribute und booleschen Operatoren der Constructive Solid Geometry mit konstanter Komplexität. Die *Modellierung* kann in die Teilbereiche *geometrisches* und *implizites* Modellieren unterteilt werden. Das *geometrische* Modellieren umfasst die lokale Gitterverfeinerung, um die numerische Auflösung eines bestehenden Gitters zu verbessern. Das *implizite* Modellieren behandelt das Erzeugen und Manipulieren von impliziten Modellen. Dazu wurde ein neuartiger Algorithmus zur Oberflächenrekonstruktion von komplexen $3d$-Flächen aus schlecht konditionierten Daten, wie sie oft in geologischen Anwendungen auftreten, entwickelt. Dabei kann die Fläche begrenzt sein und Unstetigkeiten aufweisen. Des Weiteren wurde im Rahmen dieser Arbeit eine innovative Methode zum intuitiven Echtzeit-Manipulieren von implizit definierten Objekten wie Horizonte, die auf der GeoChron-Parametrisierung basieren, erarbeitet. Die Manipulation erfolgt interaktiv direkt am $3d$-Volumenmodell wobei geologische Randbedingungen automatisch eingehalten werden.

Centre de Recherches Pétrographiques et Géochimiques
Laboratoire d'Infographie et d'Analyse de Données
Rue du Doyen Marcel Roubault - 54500 Vandœuvre